

TASK: UR20
CDRL: 00980

UR20 — Process/Environment Integration

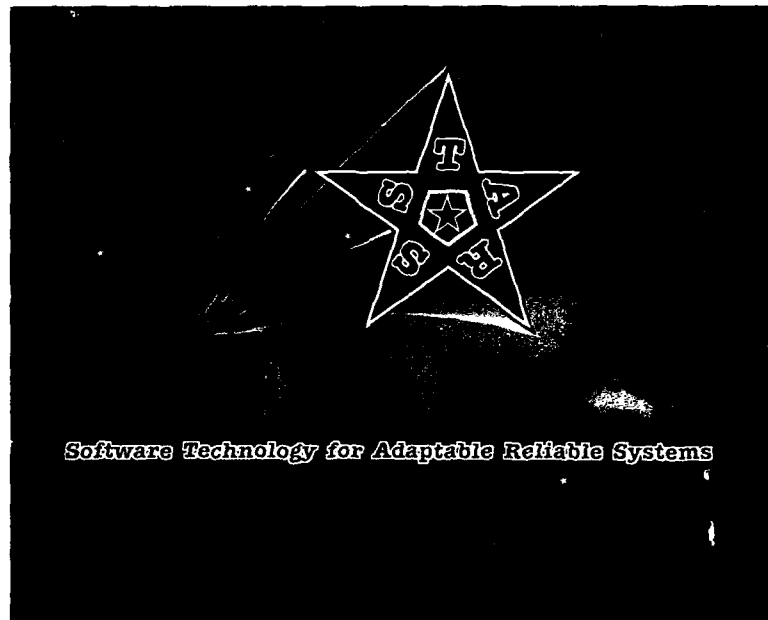
Ada Command Environment (ACE) User's Manual

Informal Technical Data

DTIC FILE COPY

UNISYS

AD-A229 400



STARS-RC-00980/001/00

29 October 1990

DTIC
ELECTE
NOV 14 1990
S B D

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

00 11 13 130

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

29 October 1990

3. REPORT TYPE AND DATES COVERED

Final

4. TITLE AND SUBTITLE

User Manual for Ada Command Environment (ACE)

5. FUNDING NUMBERS

STARS Contract
F19628-88-D-0031

6. AUTHOR(S)

William P. Loftus
John A. Thalhammer

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

8. PERFORMING ORGANIZATION
REPORT NUMBER

GR-7670-1141 (NP)

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force
Headquarters, Electronic Systems Division (AFSC)
Hanscom AFB, MA 01731-5000

10. SPONSORING/MONITORING
AGENCY REPORT NUMBER

00980

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release;
distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This technical report provides users of the Ada Command Environment (ACE) with a description of each of the packages and subprograms that are provided with ACE. In addition, the report describes the overall purpose of ACE, general guidelines on ACE usage, and provides examples of typical user interaction during an ACE session.

14. SUBJECT TERMS

Ada Command Environment (ACE)
Abstract Data Types (ADT)
X Window System

15. NUMBER OF PAGES

142

16. PRICE CODE

17. SECURITY CLASSIFICATION
OF REPORT

Unclassified

18. SECURITY CLASSIFICATION
OF THIS PAGE

Unclassified

19. SECURITY CLASSIFICATION
OF ABSTRACT

Unclassified

20. LIMITATION OF ABSTRACT

CAR

TASK: UR20
CDRL: 00980
29 October 1990

USER MANUAL
For The
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

Ada Command Environment (ACE)
Version 8.0
SunOS Implementation

STARS-RC-00980/001/00
Publication No. GR-7670-1141(NP)
29 October 1990

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0002

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

Unisys Defense Systems
Tactical Systems Division
12010 Sunrise Valley Drive
Reston, VA 22091

Distribution Limited to
U.S. Government and U.S. Government
Contractors only:
Administrative (29 October 1990)

TASK: UR20
CDRL: 00980
29 October 1990

USER MANUAL
For The
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

Ada Command Environment (ACE)
Version 8.0
SunOS Implementation

STARS-RC-00980/001/00
Publication No. GR-7670-1141(NP)
29 October 1990

Data Type: A005, Informal Technical Data

CONTRACT NO. F19628-88-D-0031
Delivery Order 0002

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

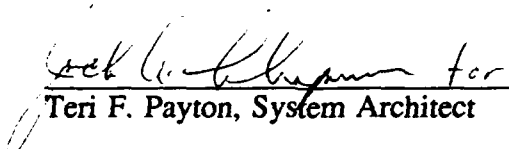
Prepared by:

Unisys Defense Systems
Tactical Systems Division
12010 Sunrise Valley Drive
Reston, VA 22091

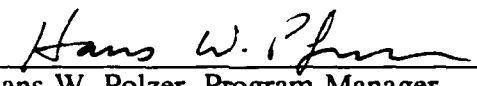
PREFACE

This document was prepared by Unisys Defense Systems, Valley Forge Laboratories, in support of the Unisys STARS Prime contract under the Process/Environment Integration task (UR20). This CDRL, 00980, is type A005 (Informal Technical Data) and is entitled "Ada Command Environment (ACE) User's Manual, Version 8.0".

Reviewed by:


Teri F. Payton, System Architect

Approved by:

 10/30/90
Hans W. Polzer, Program Manager

Approved for	
Special Agent	<input checked="checked" type="checkbox"/>
Inspector	<input type="checkbox"/>
Supervisor	<input type="checkbox"/>
per letter	
Signature	
Date	
Initials	

A-1



Contents

1	Introduction	1
1.1	Intended Audience	1
1.2	Reference Documents	1
2	The Command Language of ACE	1
2.1	The Environment of ACE	2
2.2	Interaction with ACE	3
3	Starting ACE	3
3.1	Example	4
4	Software and Hardware Environment	4
4.1	Sun Workstation	5
5	Pragmas Supported by ACE	5
6	Environment of ACE	7
6.1	ACE Libraries and Environment Tailoring	9
6.2	ADT Philosophy	12
6.2.1	The Ada Language Standard	13
6.2.2	Command Structure	14
6.2.3	Command Applicability	15
6.2.4	Command Specialization	16
6.2.5	Command Extensibility	16
6.3	ADT Interfaces within ACE	16
6.4	ADT Body Implementations	17
6.5	External Images	17
6.6	ADT Summary	18
6.7	ACE Command List	20
7	Abstract Data Types of ACE	23
7.1	Standard Packages	23
7.1.1	Objects	24
7.1.2	Standard	24
7.1.3	System	29
7.1.4	Io_Exceptions	29
7.1.5	Low_Level_Io	30
7.1.6	Calendar	30
7.1.7	Text_Io	31
7.1.8	Ace_Standard	35
7.1.9	Strings	36
7.2	Command Language Commands	36
7.2.1	Ace_Adt	37
7.2.2	Host_Os	38

7.2.3	Manipulate_Scope	38
7.2.4	Debugger	39
7.2.5	File_System	41
7.2.6	Directory_Objects	42
7.2.7	Text_Objects	44
7.2.8	Program_Objects	45
7.2.9	Program_Text_Objects	46
7.2.10	Binary_Objects	47
7.2.11	Program_Units	47
7.2.12	Help_Adt	50
7.2.13	Object_Lister	50
7.3	Key Bindings	51
7.3.1	Key_Bindings	51
7.4	Windowing Commands	56
7.4.1	Window_Objects	56
7.4.2	Ace_X_Window_System	56
7.4.3	Ace_User_X_Window_System	57
7.4.4	Window_Draw_Routines	58
7.5	CAIS-A Commands	59
7.5.1	CAIS_Routines	59
7.5.2	STARS_Tools	61
7.6	CPU Timing Package	63
7.6.1	Cpu_Time	63
7.7	Xt Toolkit Interface	63
7.7.1	X_Windows	64
7.7.2	Renamed_Xlib_Types	65
7.7.3	Intrinsics	66
7.7.4	Widget_Package	67
7.7.5	Hp_Widgets	84
8	More Information About Some Ace Features	109
8.1	Xt Toolkit Interface	109
8.1.1	Xt Prototyping Sessions	110
8.1.1.1	Starting an Xt Prototyping Session.	110
8.1.1.2	Finishing an Xt Prototyping Session	111
8.1.2	Xt Argument Lists	112
8.1.3	Xt Callbacks	113
8.1.3.1	Callback Procedures.	113
8.1.3.2	Action Procedures.	114
8.1.3.3	Callback Interpretation.	115
8.1.3.4	Prototyping Callbacks.	115
8.1.4	Transition to Compiled Code	116
8.1.5	A Small Example	117
8.2	The <i>Key_Bindings</i> Package	119
8.2.1	Making Bindings	119

8.2.1.1	Using <i>Interpret_String</i>	121
8.2.1.2	Internixing I/O with <i>Key_Bindings</i> Routines.	121
8.2.2	An Example for <i>Interpret_String</i>	122
9	Supported Ada Features	123
10	Examples	130
10.1	Interactive Ada Example	130
10.2	Manipulating Ada Components	131
10.3	Interfacing with Host O/S (UNIX)	134

1 Introduction

The Ada Command Environment (ACE) is an interactive command language environment for Ada software development. Ada is both the programming language and the command language used within ACE. The paradigm and philosophy used by the Ada programmer during program development are extended into the environment in which program development takes place. ACE is modeled after other interactive programming environments, such as Smalltalk and Interlisp, which are touted for individual programmer productivity. Whereas other interactive programming environments are targeted toward *programming-in-the-small*, ACE supports *programming-in-the-large* techniques that are key elements of the Ada language.

1.1 Intended Audience

This document assumes the user has a basic understanding of the Ada language, including its concepts and the use of Ada for software design and development. This document is not tutorial in nature with regards to the Ada language. The user is directed to one of the many texts on Ada or the Ada Language Reference Manual for background on Ada.

1.2 Reference Documents

MIL-STD-1815A *Ada Programming Language*, 22 Jan 1983

William P. Loftus, Charles L. Oei, and John A. Thalhamer. The Ada Command Environment—ACE. In *Proceedings of Ada Expo '88*, Anaheim, California, October 1988.

John A. Thalhamer, William P. Loftus, Charles L. Oei, Ralph A. Foy. Ada Abstract Data Types—the Foundation of an Interactive Ada Command Environment. *Proceedings of the Seventh Annual National Conference on Ada Technology*, Atlantic City, New Jersey, March 1989

Unisys, *Ada Command Environment Installation Guide*, Version 8.0, Informal Report, U.S. Department of Defense Contract No. F19628-88-D-0031, 12 April 1990

2 The Command Language of ACE

The command language accepted by ACE is Ada. Within ACE, Ada is used as the language for program development as well as the mechanism for traditional interaction with the host operating system. Each command entered into ACE must be a legal Ada construct.

Ada as a command language allows interactive program development, typical user interaction with the host environment, and the development of command language procedures. Whereas within other environments different language constructs are needed for the com-

mand language and command language procedures (not to mention the different programming language). ACE allows Ada to be used as the unifying language.

ACE immediately executes Ada constructs as they are presented. The set of Ada constructs that are immediately executed are compilation units, statements, and basic declarations. Upon the completion of one of the constructs, the statements associated with the construct are executed and the appropriate results are given based upon the Ada statements that were evaluated.

Immediate execution of the Ada constructs for compilation units, statements, and basic declarations allows Ada to be effectively used as a command language. An interactive development environment requires a dynamic atmosphere in which the next operation performed may be based upon a previous result. The ability to intermingle declarations, statements, and compilation units as commands to ACE is the basis for a dynamic environment. For example, a typical order of Ada commands to ACE may be the declaration of an object, assignment of an initial value to that object, the definition of a subprogram specification and body, followed by an invocation of the subprogram using the declared object. This sample shows the ability to intermix the sequence of declarations, statements, and compilation units that are submitted to ACE.

2.1 The Environment of ACE

ACE provides a basic set of Ada objects and operations as the initial environment of ACE. These objects and operations are encapsulated as abstract data types (ADTs) and implemented as a set of Ada packages. Upon initialization of ACE, the basic set of ADTs are assimilated into a base environment for the user. This base environment includes packages that are necessary for the interpretation of Ada statements (such as the Ada package *Standard*), as well as packages that provide operations typically performed by a user when interacting with the underlying operating system.

Each subprogram or package that is entered by the user during an ACE session may be viewed as an extension of the environment. The subprogram and/or package provides a set of objects and operations which is available to the user. These subprograms and packages may be permanently stored so that they will be persistent between ACE sessions. Upon start-up of the user's subsequent ACE session, these user-defined ADTs may be included into the ACE environment, thus tailoring the ACE environment to the user's preferences.

Common, every day operations that would typically be invoked by the majority of ACE users are made directly visible to the user as a default. The user may simply acquire direct visibility of other ADTs through the Ada *use* statement. In the event that direct visibility is not desired when given as a default, ACE provides a set of ADTs that may "undo" the Ada *use* statement. More detail on the operation of this type of ADT is provided in the description of the dynamic environment ADTs.

The description of the ADTs that are provided as the base environment for ACE are described

later in this manual (see section 7).

2.2 Interaction with ACE

The user is provided with a mechanism for controlling the operating characteristics of the Ada Command Environment. Within Ada, the *pragma* construct is the means for issuing directives to the compiler which do not affect the legality of the Ada program. Since ACE is based upon the paradigms of Ada, the Ada *pragma* construct is utilized within ACE to control the operation of ACE.

In addition to the *pragma* directives of ACE, Ada objects and operations (in the form of an ADT) are employed to provide user control over ACE's input and output mechanisms. The Ada *Text_IO* package contains the definition of a set of operations to control the default input and output files associated with the Ada program. A similar facility is provided within ACE to control the default input and output files associated with ACE. For example, *Standard_Ace_Input* and *Standard_Ace_Output* are defined to return a file type associated with the keyboard and CRT, respectively. Additional operations such as *Set_Ace_Input* and *Current_Ace_Input* are used to modify and acquire the current file type which defines the file from which ACE is to acquire input. Similarly named operations exist for output.

A description of the *pragmas* supported by ACE and the ADTs associated with the operation of ACE are given later in this manual (see section 5).

The Exit Statement

The *exit* statement is used to terminate the execution of subprograms within ACE or ACE itself.

- *exit Ace* terminates the execution of ACE and returns the user to the host operating system.
- *exit Ace_Level* terminates the current level of execution and returns the user to the next higher level.
- *exit Ace_Main* terminates execution of all subprograms and returns the user to the top level of ACE.

3 Starting ACE

ACE can be started by typing the command "ACE" at the Unix prompt. This initiates the ACE session. During start up, ACE processes the definitions of the default abstract data types that will be available during the ACE session. A noticeable delay occurs during the

initiation of ACE while this set of abstract data types is processed. ACE is ready to accept user commands after the display of the ACE main prompt—"ACE> ".

When ACE has begun accepting a command and requires more information to complete the command, ACE will display a continuation prompt " |". This prompt indicates that the Ada construct being entered as a command is not yet complete. Examples of these Ada constructs include a compound statement, such as an *if* statement, a subprogram body, or even an assignment statement or declaration statement that is entered on multiple lines. The continuation prompt indicates that ACE is awaiting further input associated with the incomplete command, and will execute the command upon completion of the command.

3.1 Example

```
ACE>
ACE> i : integer;
ACE> i := 100;
ACE>
ACE> if i < 0 then
|   put_line ("negative value");
| elsif i > 0 then
|   put_line ("positive value");
| else
|   put_line ("zero value");
| end if;
positive value
ACE>
```

4 Software and Hardware Environment

The ACE prototype is operational on a Sun-3 workstation. The description of each of these environments is detailed below.

Execution of ACE should be performed within the same directory in which the executable image of ACE resides. The files that should be located within this directory are as follows:

- *ACE*—the ACE executable image
- *startup.ace*—the environment initialization file processed by ACE when ACE is started
- *ace_ada*—script that interfaces with the host Ada compilation system (Verdix on the Sun workstation). This may be tailored to local preferences.
- *ace_edit*—script that interfaces with the host editor (*vi* on the Sun workstation). This may be tailored to local preferences.

- *observe_window.icn*—iconic image for the windows created by pragma *observe*, which is only needed when running within a window environment and when a window is closed to its iconic form.

4.1 Sun Workstation

Within the Sun workstation environment, ACE provides a simple interface to the X Window System. However, ACE may be executed independently without a supporting windowing system. Window manipulation operations are provided with the X Window System—ACE has integrated Xlib-Ada binding from the Unisys UR20 user interface task to provide a programmatic interface to X.

To execute ACE, the suggested configuration is a Sun-3 workstation running:

- SunOS 3.5
- X Window System, X 11 Release 3, if running under X
- Verdex Ada Development System version 5.5 and the *vi* editor

To create an ACE executable image, the suggested configuration is a Sun-3 workstation running:

- SunOS 3.5
- X Window System, X 11 Release 3, if using a window system
- Verdex Ada Development System version 5.5
- Ada bindings to Xlib (Available from Unisys STARSCenter)
- C compiler provided with SunOS 3.5

5 Pragmas Supported by ACE

Pragmas are the mechanism for controlling ACE's operating characteristics. These pragmas control the production and format of information produced by ACE, and are particularly useful for debugging purposes. The following pragmas are supported in ACE:

1. *Observe*

Takes an enumeration string literal (*Objects*, *Statements*, or *Subprograms*) as the first argument and one of the identifiers *On* or *Off* as the second argument. Default at ACE initialization is *Off* for all three observation items.

An observation window is created to observe the manipulation of objects, statements, or subprograms. *On* enables observation; *Off* disables observation. *Objects* displays the definition of each object within ACE when it is elaborated. *Statements* displays each Ada statement as it is executed within ACE. *Subprograms* displays each Ada subprogram as it is executed, by displaying the Ada statements that make up the subprogram body and highlighting the statement number of each statement as the statement is executed. (Execution speed of ACE is artificially decreased during *Subprograms* observation to allow appropriate visual recognition of the trace of statement execution.)

Only a single *Subprogram* observation window is created. For *Objects* and *Statements*, each pragma *Observe* with *On* will create a new observation window, and will not delete the previous observation window. This allows tracing and comparison of several program execution paths.

2. *Echo*

Takes one of the identifiers *On* or *Off* as the single argument. Default at ACE initialization is *Off*.

Echo will display within the primary ACE window each line of input that is being processed by ACE.

3. *Dump*

Takes one of the identifiers *On* or *Off* as the single argument. Default at ACE initialization is *Off*.

Dump will display the Ada statement that has been syntactically and semantically checked by ACE prior to its execution.

4. *Trace*

Takes one of the identifiers *On* or *Off* as the single argument. Default at ACE initialization is *Off*.

Trace will display within the primary ACE window each Ada statement as it is executed. (This is similar to *pragma Observe (Statements, On)* except that tracing output is sent to the primary ACE window rather than to a statement observation window.)

5. *Debug*

Takes one of the identifiers *On* or *Off* as the single argument. Default at ACE initialization is *Off*.

Debug will force ACE to save the local symbolic information associated with subprograms during subprogram execution. This allows local symbolic information to be referenced when breakpoints are triggered within subprograms. This saving of local symbolic information decreases the execution speed of ACE.

6. *List_Statement_Numbers*

Takes one of the identifiers *On* or *Off* as the single argument. Default at ACE initialization is *Off*.

List_Statement_Numbers will flag the internal *Dump* routines to associate numbers with the statements that are dumped. These numbers can be used in association with the ACE Debugger ADT.

7. *Main_Prompt*

Takes a string as its only argument.

Main_Prompt will set the main prompt of ACE.

8. *Continue_Prompt*

Takes a string as its only argument.

Continue_Prompt will set the continuation prompt of ACE.

9. *Builtin*

FOR DEVELOPERS/SYSTEM ADMINISTRATORS ONLY

Takes a string literal, the name of a subprogram, as the first argument and a static expression of the predefined integer type as the second argument.

The body associated with the identified subprogram specification is built-in to ACE. The body has been pre-compiled and merged into ACE. ACE does not expect further Ada input to define the body for this built-in subprogram. The second argument is an internal numbering scheme within ACE to uniquely identify each built-in subprogram.

6 Environment of ACE

The environment encompasses the traditional view of programmer interaction with the operating system, such as file manipulations, editors, compilers, debuggers, and windowing systems. A set of consistent Ada packages has been constructed to encapsulate these traditional objects and operations. Moreover, the environment extends the user's view to include interaction with parts of Ada software systems to facilitate *programming-in-the-large* activities.

The environment consists of a set of Ada packages that is interpreted by the ACE Ada interpreter, where each package defines an ADT which the user may manipulate, redefine, or extend. ADTs implemented in ACE which support the user's traditional view of an environment include:

- a help ADT
- a host operating system ADT
- a hierarchical typed file system ADT (which includes the editor, compiler, stubber, and other file support tools)

- a windowing ADT
- a symbolic debugger ADT
- Ada predefined library packages (Standard ADT, ASCII ADT, System ADT, Text_Io ADT, Io_Exceptions ADT, Low_Level_Io ADT, Calendar ADT, Object List ADT)

ADTs implemented in ACE which support rapid prototyping and *programming-in-the-large* include a package ADT and subprogram ADT (i.e. abstract data types that allow Ada packages and subprograms to be manipulated as data).

For example, the ACE user could enter a set of Ada procedures into the Ada interpreter, testing and refining them as needed. Then, using the package and subprogram ADTs, the user could define a package and insert the procedures into the package, thus creating a new ADT. This illustrates the power ACE provides in extending the environment, and the ease by which the user may accomplish it. ACE provides the mechanism for the user to interactively define a set of data types and operations on the data types. ACE also allows the user to further manipulate these types and operations by encapsulating them into abstract data types. Moreover, the user may then collect a set of ADTs into a software system.

Currently, the ACE-developed ADTs provide basic, low-level functionality within their particular domains. This avoids biasing the ACE environment towards specific existing environments. Users can easily tailor or augment the basic routines to personal or project needs. As general-purpose higher levels of abstraction are defined, additional ADTs will be developed which extend the ACE environment. The basic ADTs serve as building blocks for the next higher level of abstraction.

Use of Form Parameters in ACE

The *Form* parameter is used within Ada Input-Output (Chapter 14 of the Ada Reference Manual) to give system-dependent characteristics that may be associated with a file. For example, the *Form* parameter of the *Open* subprogram in the package *Text_Io* is used to define attributes of the specified file (e.g., protection, type of file, etc.). Since ACE supports *Text_Io* (and other standard Ada packages) the *Form* parameter is also supported.

In addition, the ACE Command Language uses the *Form* parameter as a mechanism within other abstract data types to give system-dependent characteristics that are associated with the objects being manipulated within the package. Some ADTs within ACE are concerned with the manipulation of subprograms. Expanded names alone cannot uniquely identify overloaded subprograms. The *Form* parameter is the mechanism by which overloaded subprograms are uniquely identified. Each Ada statement within ACE is assigned a unique statement number. This statement number, passed as a string in the *Form* parameter, may uniquely identify the appropriate subprogram.

In order to acquire the statement number (or *Form*) attribute of a subprogram, ACE also

provides a *Form* function within several ADTs. When invoked upon a subprogram or package object, this function will return the string *Form* of the respective object.

The following example will highlight the use of the *Form* parameter. *List* is an operation that can display a subprogram's specification and/or implementation. In the example, invocations of *List* illustrate acquisition of the most recently defined subprogram of a given name (*Interpret*), all subprograms of the same given name, or a specific subprogram of the given name—identified via the *Form* parameter.

```
ACE> List ("interpret");
```

```
procedure interpret (file : in file_type; error : out boolean);
-- Form => " 377"
pragma builtin (interpret, 505);
```

```
ACE> List ("interpret", "all");
```

```
procedure interpret (file : in file_type; error : out boolean);
-- Form => " 377"
pragma builtin (interpret, 505);
```

```
procedure interpret (str : in string; error : out boolean);
-- Form => " 375"
pragma builtin (interpret, 504);
```

```
procedure interpret (tree : in ace_statement_database; error : out boolean);
-- Form => " 373"
pragma builtin (interpret, 503);
```

```
ACE> List ("interpret", "375");
```

```
--? interpret will "execute" the value of the str parameter.
procedure interpret (str : in string; error : out boolean);
-- Form => " 375"
pragma builtin (interpret, 504);
```

6.1 ACE Libraries and Environment Tailoring

When ACE is executed, an initialization file is automatically *interpreted*. This initialization file is called *startup.ace* and is located in the user's home directory. This file contains ACE

commands (in the form of Ada) that define all the operations (i.e., subprograms) that will be recognized by ACE. The following is an example of a startup.ace file:

```
--pragma echo(on);
--pragma trace(on);

Demo_Directory      : constant String := "/ace/demo/";
Startup_Directory   : constant String := "/ace/startups/";

-- Define "&", so we can use environment strings (e.g., Demo_Directory).
function "&" (Lhs, Rhs : String) return String;
pragma builtin("&", 0);

-- get routines for measuring CPU.
Interpret_File (Startup_Directory & "cpu_time.ace");

-- Variables for clocking our startup speed.
Start : Time;
Stop  : Time;

-- Start ticking
Start := Clock;

-- Ada's Standard Package
Interpret_File (Startup_Directory & "standard.ace");

-- Key Mappings
Interpret_File (Startup_Directory & "bindings.ace");

-- Normal Command Language commands (e.g., Set_Directory)
Interpret_File (Startup_Directory & "commands.ace");

-- Windowing ADTs
Interpret_File (Startup_Directory & "windowing.ace");

-- Some developer debugging aids
Interpret_File (Startup_Directory & "developer.ace");

-- Xt toolkit ADTs
Interpret_File (Startup_Directory & "xt.ace");

-- Xt demonstration
Set_Directory (Demo_Directory & "Xt");
--Console ("Support");
--Interpret_File ("support");
```

```
--Console ("Globals");
--Interpret_File ("globals");
--Console ("Edit_panel");
--Interpret_File ("edit_panel");
--Console ("Callbacks");
--Interpret_File ("callbacks");
--Console ("Ace.buttons");
--Interpret_File ("ace.buttons");
--Console ("Demo");
--Interpret_File ("demo");
--New_Line;
```

procedure Back_Word is

begin

 Do_Move_Left;

 while Key_Bindings.Do_Get_Current_Character = ' ' and
 Key_Bindings.Do_Get_Current_Column /= 1 loop

 Do_Move_Left;

 end loop;

 while Key_Bindings.Do_Get_Current_Character /= ' '
 and Key_Bindings.Do_Get_Current_Column /= 1 loop

 Do_Move_Left;

 end loop;

 if Key_Bindings.Do_Get_Current_Column /= 1 then

 Do_Move_Right;

 end if;

end Back_Word;

procedure Kill_Word is

begin

 Back_Word;

 while (Do_Get_Current_Character /= ' ' and

 Do_Get_Current_Character /= Ascii.Nul) loop

 Do_Delete_This_Char;

 end loop;

 Do_Delete_This_Char;

end Kill_Word;

procedure Forward_Word is

begin

 while (Do_Get_Current_Character /= ' ' and

 Do_Get_Current_Character /= Ascii.Nul) loop

 Do_Move_Right;

 end loop;

 while (Do_Get_Current_Character = ' ') loop

```

        Do_Move_Right;
    end loop;
end;

-- Set some history and editing key commands
Make_Binding (Ascii.Eot, Delete_This_Char);
Make_Binding (Ascii.Nak, Kill_Line);
Make_Binding (Ascii.Enq, Key_Bindings.End_Of_Line);
Make_Binding (Ascii.Soh, Beginning_Of_Line);
Make_Binding (Ascii.Dc2, Refresh_Current_Line_And_Prompt);

Make_Binding (Ascii.Etb, Interpret_String, "Kill_Word;");
Make_Binding (Ascii.Ack, Interpret_String, "Forward_Word;");
Make_Binding (Ascii.Stx, Interpret_String, "Back_Word;");

Make_Binding (Ascii.Esc & "[A", History_Back);
Make_Binding (Ascii.Esc & "[B", History_Forward);
Make_Binding (Ascii.Esc & "[D", Move_Left);
Make_Binding (Ascii.Esc & "[C", Move_Right);

-- Stop Ticking
Stop := Clock;

-- How much time?
Put ("Startup CPU seconds: ");
Put_Time(Difference(Stop, Start));

-- ASCII Terminal clear to EOL.
Put_Line(Ascii.Esc & "[K");

```

The *Interpret_File* command is predefined in ACE, and can be used without being defined by the user. All other commands and variables must be defined by a previous declaration. Through the editing of this file the user can customize and tailor their environment. The ACE libraries (.ace files) referred to in the example are described in section 7.

6.2 ADT Philosophy

Data abstraction, information hiding, modularity, and locality are some of the the modern software engineering principles used in the development of software applications. The notion of data abstraction is also a powerful mechanism for the definition of a command environment—an environment that contains a set of objects upon which a group of command operations act.

An *abstract data type* is an abstraction mechanism that encapsulates a set of values together

with a set of operations that apply to the values. Within software development, the decomposition of the system may be defined through a set of objects, the operations applicable to the objects, and the operations needed by the objects. ADTs serve as a natural description method for this type of system decomposition. ADTs are also a key component of the object-oriented design and development approach.

The directives issued by a software developer to the underlying host environment may also be naturally defined through the use of ADTs. Each directive or command may be viewed as an operation; the qualifiers or parameters may be viewed as the objects upon which the operation is performed. Logically associated objects and operations may be gathered together into collections which are related to particular components of the underlying host environment. Thus, a parallel can be drawn between abstract data types and the composition of a command language.

Many of the newer procedural languages provide syntactic mechanisms to easily specify and manipulate ADTs. Ada is one such language. The constructs of packages (specification and body), subprograms (functions and procedures), subprogram invocation, type declarations, object declarations, and context clauses are examples of Ada's support for ADTs. The Ada Command Environment makes use of these Ada constructs to define the environment objects and operations through ADTs.

ACE provides an Ada ADT interface to the underlying host environment in the form of Ada package specifications. The package specifications are processed by ACE upon initiation. Thus, a set of predefined types and operations are made available to the user from the beginning of an ACE session. Since these types and operations are defined via the Ada package construct, the methods used to manipulate Ada packages are also used to manipulate the operation of the environment ADTs.

6.2.1 The Ada Language Standard

Ada, as a modern procedural language, encompasses many of the state-of-the-art software engineering principles. These principles are extended into the command environment through the use of Ada to define the environment with ADTs.

The Ada package construct supports the principles of data abstraction and information hiding through the separation of the package specification from the package body. The separation of the specification and implementation of the abstract data type in Ada and ACE is a key element in the ability to tailor the environment. Different implementations of an environment ADT specification are an obvious mechanism for tailoring the environment to a project's taste. For example, a common configuration management interface may be defined through a single ADT specification, but different implementations may be written based upon the project's particular selection of a configuration management application system.

The ability to layer ADTs within Ada supports the principles of modularity and locality.

Environment extensibility may be accomplished through the use of layered ADTs. For example, a new ADT specification may be written that presents an interface that is more familiar or comfortable to the user. The implementation of that ADT simply invokes the standard set of operations. The ADT makes the translation from user orientation to system orientation, rather than forcing the human to mentally perform the translation. Layered ADTs also support the notion of different levels of abstraction. For example, the notion of formatting a textual document, building its table of contents, and printing the result on a printer may be viewed as either a single operation or a series of lower level operations. Low level ADTs serve as the building blocks for higher level ADTs.

Within the language definition of Ada, Ada is used to extend its own definition. The Ada input-output operations (chapter 14 of the reference manual) are provided in the language by the means of predefined packages. In addition, other predefined library packages are required for each Ada implementation. ACE has implemented the Ada predefined packages, such as *Standard*, *ASCII*, *Calendar*, *System*, and *Text_IO*. This set of packages makes the standard Ada types and operations available in the command environment. Continuity is established between the command environment and the typical Ada development environment.

ACE also views the set of Ada predefined packages defined in the reference manual as a set of guidelines to be followed in the development of environment ADTs. The input-output packages of chapter 14 of the reference manual denote a style of operation definition and object manipulation that ACE has expanded to encapsulate the entire environment. The *Create*, *Open*, *Close*, and *Delete* procedures that are applicable to file objects are used within the command environment to define similar control operations upon other types of objects. An example of this is the similar treatment of file objects and window objects. File objects and window objects are each abstract data types in ACE that are created using the *Create* procedure and removed using the *Delete* procedure. The operations that the Ada developer is familiar with in the program development environment are the same operations that are to be invoked within the host environment to accomplish similar tasks.

The guidelines are followed in more detail than simply through subprogram names. Names and modes of parameters, the selection of a procedure versus a function, and the use of the *Form* parameter as a string data type to specify non-default implementation options are all further examples of following the style of Ada as defined in the language standard. These and other instances of conformance within ACE, enforce an Ada-oriented style of ADTs within the ACE environment.

6.2.2 Command Structure

Consistency and uniformity in the command environment of ACE is achieved through the use of Ada and ADTs. Commands and objects are logically grouped together as ADTs via the Ada package mechanisms. This grouping allows the environment to be structured and ordered. In addition, by nesting packages and subprograms the environment provides controlled access to information. Users explore the environment in an orderly and informative manner. This logical grouping of environment components has many benefits over the flat

structure supported by most command languages.

For example, if a specific windowing package is nested inside a basic windowing package, novice users must "use" or reference the basic windowing package before they can access the specific windowing package. This does not guarantee that novice users understand the environment. However, it does guarantee that novice users understand the logical structure of the environment. Of course, expert users who know the structure of the environment are not hindered, since they can simply reference an arbitrarily nested command via the Ada expanded name feature.

Another benefit of this command structure combined with Ada is the ability to define a user interface that is consistent with the paradigms of Ada, as well as uniform in its treatment of objects and operations in the environment. Such an environment would support (at all levels of interaction with the environment) Ada philosophies, providing an excellent vehicle for Ada development. The facilities of overloading and derived subprograms in Ada provide the opportunity to define uniform interfaces to logically related operations and objects. As described above, the ability to define a Create operation for each type of environment object is supported in Ada through overloading. ACE supports overloading to allow the uniform definition of abstract data types across the entire command environment. In addition to being consistent with the Ada standard, the environment is also uniform among the ADTs that are defined within it.

6.2.3 Command Applicability

One benefit of modern procedural languages is the notion of strong typing. The benefits of strong typing within Ada are also of benefit to Ada as a command language and the definition of ADTs. While ADTs allow the definition of operations for objects, strong typing enforces the proper use of the operations. Many of the problems associated with a novice's use of a command language can be attributed to the application of operations to inappropriate objects (e.g., printing a binary image). In a strongly typed command language, and in particular ACE, if there is no operation "print" defined for binary image objects then the user can not (even accidentally) apply the operation.

Another benefit of strong typing in a command language is in the operation of very large software systems. Many of the benefits of using ADTs in the construction of these software systems are retained in the command language which acts as the "glue" which holds such systems together. Having a strongly typed command language helps guarantee that the systems are correctly constructed from their components. In addition, having a compilable command language allows an interpreted system to become an entirely compiled system merely by compiling the command language, whereas in a traditional command language, the "glue" would have to be rewritten into the system's programming language.

6.2.4 Command Specialization

Through the use of derived types and derived subprograms, new objects can be described as specializations of existing objects, i.e., described as differences from existing objects. For example, the entire abstract data type for ACE's hierarchical file system is constructed of existing ADTs that are specializations of a general file ADT. The general file ADT provides the basic operations (e.g., *Create*, *Delete*, *Copy*, *Rename*, etc.) that can be performed on all files. The immediate specializations of the general file ADT are *Text_Files*, *Directory_Files*, and *Binary_Files*. Each of these specializations provides specific new or redefined operations for each type. Any operation defined for the general file ADT that is not redefined in a specialization's operations is inherited by the specialization. Therefore, each specialization of the general file ADT inherits the *Create*, *Delete*, etc. operations, which in turn allows every type of file in the file system to be manipulated via the general file operations. Specialization provides a very powerful reuse mechanism within ACE; existing objects can be extended or tailored for particular applications or user aesthetics without having to describe the entire ADT.

In addition, since Ada (and consequently ACE) implicitly derives subprograms for every derived type, much of the work that is normally associated with strong typing in a command language and the construction of a hierarchical command environment is removed from the user. Each derived type implicitly inherits a set of commands that enable its basic manipulation.

6.2.5 Command Extensibility

An important part of any state-of-the-art environment is the ability of the environment to evolve as technology and methodologies evolve. ACE's approach is to use Ada ADTs to define the command language (creating a command environment). As described before, Ada ADTs have a clean separation of implementation from specification. Therefore, as technology makes small leaps, the new techniques can be incorporated in the ADT implementation while not effecting the specification. In addition, when radical breakthroughs are made in technology, new environment ADTs can be constructed and incorporated into the command environment. Using this approach, we are only limited by the ability of Ada to assimilate new approaches.

6.3 ADT Interfaces within ACE

Abstract data types within ACE are defined by Ada packages. The package specifications encapsulate the definition of the objects and the operations that are applicable to the objects. Additionally, the package specification provides a mechanism for information hiding, particularly hiding of the operations' implementations. The Ada package body contains the implementation of the object and its respective operations.

ACE supports two mechanisms for the implementation of the ADT bodies: *interpreted* and

built-in. Each of these mechanisms supports a different facet of environment definition, and together they provide the facilities to compose and extend the Ada command environment. Additionally, ACE through its ADTs provides a mechanism to access executable images external to ACE. This provides added power and flexibility to the command environment.

6.4 ADT Body Implementations

As previously stated, Ada is the command language accepted by ACE and interpreted by ACE's command language interpreter. The environment (as defined by Ada packages) is read by the command language interpreter and processed, resulting in the elaboration of Ada packages. This process of interpreting Ada ADT package specifications and bodies is the typical method through which ADTs are declared within ACE.

ACE provides an additional mechanism by which package bodies may be defined. Rather than interpreting an Ada package body, the Ada code may be compiled and linked into the ACE executable. The package specification for the package is still Ada code that is interpreted by ACE. A pragma directive informs ACE that the package body associated with this package specification is already compiled and included within ACE.

This method of package body inclusion provides benefits to the runtime efficiency of ACE. ACE may be tuned such that frequently invoked code is executed at the machine language level (i.e., the compiled level), rather than interpreted.

Another benefit of compiled implementation is that it provides interactive invocation and composition of compiled code within the command environment. An example of this is the X Window System ADT of ACE which provides Ada interfaces to the X Window System (currently implemented in C).

6.5 External Images

A vast array of applications and support tools are typically available within the host environment. ACE does not impose a restrictive environment that limits the facilities available to the software developer. Through a host operating system ADT, ACE provides an interface mechanism which makes external executable images on the host system available from within the command environment. Thus, environment ADT specifications are able to provide the user with a consistent Ada paradigm that may interface with a diverse set of Ada and non-Ada external images, including the host operating system.

The ability to access external images provides the opportunity to build high level Ada abstractions from low level non-Ada applications. Relationships may be formed among stand-alone applications, providing a higher level data abstraction that encompasses the user's desired functionality. The intricacies and/or idiosyncrasies of the individual applications are hidden from the user in the ADT implementation. The implementation also hides the handling of intermediate results being passed between applications. The user simply sees

the specification, which is designed to provide a consistent interface within the Ada-oriented environment.

By invoking external images through environment ADTs, the functionality of ACE can be extended into domains which can be tailored to specific environments, projects, or users. For example, a project-oriented configuration management ADT can be defined which provides software configuration control objects and operations. The programs which must be accessed to support these facilities may exist scattered about the file system, or perhaps in a common directory with many other programs unrelated to configuration management tasks. The configuration management ADT can provide a coherent view of these operations and hide the organization or disorganization of the underlying programs.

6.6 ADT Summary

This section lists the ADTs which are defined in the *startup.ace* file provided with the current release of ACE and gives a brief description of the subprograms and objects provided by the package.

1. *Objects* Contains the basic definitions for all objects in the environment.
2. *Standard* Predefined identifiers based upon the package *Standard* in the LRM (C).
3. *ASCII* Contains identifiers for characters in the ASCII character set, as defined in the LRM (C). This package is defined within the package *Standard*.
4. *System* Contains identifiers for configuration-dependent characteristics. It is based upon the package *System* as defined in the LRM (13.7).
5. *Io_Exceptions* Not currently supported.
6. *Low_Level_Io* Not currently supported.
7. *Calendar* Provides the user operations on the clock, as defined in the LRM (9.6).
8. *Text_Io*
Provides facilities for input and output in human-readable form, as defined in the LRM (14.3).
9. *Ace_Integer_Io* Provides input/output functions for integers. This is a hand-instantiated version of the *Integer_Io* package defined in the LRM (14.3.7), tailored for *Integer_Io* type.
10. *Ace_Standard* Provides assistance objects and operations that are standard for ACE.
11. *Ace_Io* Provides operations that control the input/output of the ACLI.
12. *Strings* Provides operations that mimic the slicing and indexing of arrays for *Ace_String*.

13. *Ace_Adt* Provides objects and operations to support a programmatic interface to the ACLI.
14. *Host_Os* Provides an interface to the underlying host operating system.
15. *Manipulate_Scope* Provides operations to support the dynamic removal and hiding of objects from the environment.
16. *Debugger* Provides operations to manipulate the symbolic execution of subroutines.
17. *File_System* Defines objects and operations that may be performed on the hierarchical, typed file system.
18. *Directory_Objects* Provides operations that may be performed upon directories.
19. *Text_Objects* Provides operations that may be performed upon text files.
20. *Program_Objects* Provides operations that may be performed upon programs.
21. *Program_Text_Objects* Provides an interface to the the Ada Repository Stubber program.
22. *Binary_Objects* Provides operations that may be performed upon binary files.
23. *Program_Units* Provides the objects and operation for ACE compilation units.
24. *Help_Adt* Defines operations which provide the user with on-line assistance for declared objects in ACE, namely: packages, subprograms, and types.
25. *Object_Lister* Provides a routine to allow queries on the statement database for groups of objects declared the same way (i.e, all objects, all types, etc).
26. *Key_Bindings* Provides the operations that allow sequences of key strokes to be bound to editing and history functions.
27. *Window_Objects* Defines the objects that are associated with the Window ADTs.
28. *Ace_X_Window_System* Provides an interface to a small subset of the X Window system based upon the X Window naming conventions.
29. *Ace_User_X_Window_System* Provides an interface to a small subset of the X Window system based upon Ada paradigms.
30. *Window_Draw_Routines* Provides a simple set of drawing operations that may be performed in Window objects.
31. *Line_Counter* Provides a completely interpreted Ada line counter operation.
32. *Cais_Routines* Provides access to the underlying CAIS-A operations.
33. *Stars_Tools* Provides access to various STARS developed tools.

- 34. *Cpu_Time* Provides operations to determine CPU use.
- 35. *X_Windows* Provides declarations of the basic X library data types needed to use the ACE interface to the Xt toolkit.
- 36. *Renamed_Xlib_Types* Defines the connection between some type names used by Xt routines and the equivalent type names in the basic X library.
- 37. *Intrinsics* Contains the type declarations common to all Xt toolkit routines.
- 38. *Widget_Package* Provides a sample selection of Xt toolkit procedures.
- 39. *Xt_Stringdefs* Defines commonly used Xt resource name constants.
- 40. *Hp_Widgets* Provides the ACE interface to the Hewlett-Packard widget set.

6.7 ACE Command List

Ace Adt: Compile, Delete, Interpret, Interpret_File

Ace Io: Current_Ace_Input, Current_Ace_Output, Set_Ace_Input, Set_Ace_Output,
Standard_Ace_Input, Standard_Ace_Output

Ace User X Window System: Create, Delete

Ace X Window System: Clear_Window, Create_Window, Destroy_Window

Ada Standard Commands: abs, and, mod, not, or, rem, xor, +, -, <, <=, =, >,
>=, /=, *, /, **, &

Binary Objects: Execute

Calendar: Clock, Day, Month, Seconds, Split, Time_Of, Year, +, -, <, <=, >, >=

Cais Routines: Append_To_Cais_Arg_List, Create_Cais_Argument_List,
Invoke_Process, Put_Cais_File_Node_Host_Name,
Put_Cais_Node_Relationships, Put_Current_Cais_Node,
Prefix_To_Cais_Arg_List, Set_Current_Cais_Node, Spawn_Process,

Cpu Time: Clock, Difference, Put_Time

Debugger: Break, Clear_Break, Continue, Display, Display_Current_Statement,
Display_Next, Display_Previous, List, List_Breakpoints,
List_Symbol_Table, Set_Break, Step

Directory Objects: Close, Create, Current_Directory, Form, Home_Directory,

Is_Open, List, Name, Open, Put_Current_Directory, Set_Directory

File System: Copy, Create, Delete_File, Exists, Open, Rename, Reset,
Temporary_Name

Help: Help

Host: Host

Hp Widgets: Xw_Arrow_Widget_Class, Xw_Ascii_Sink_Create,
Xw_Bulletin_Board_Widget_Class,
Xw_Bulletin_Widget_Class, Xw_Button_Widget_Class,
Xw_Cascade_Widget_Class, Xw_Disk_Source_Create,
Xw_Disk_Source_Destroy, Xw_Form_Widget_Class,
Xw_Image_Edit_Widget_Class, Xw_List_Widget_Class,
Xw_Listrow_Col_Widget_Class, Xw_Manager_Widget_Class,
Xw_Menu_Button_Widget_Class, Xw_Menu_Sep_Widget_Class,
Xw_Menubutton_Widget_Class, Xw_Menumgr_Widget_Class,
Xw_Menupane_Widget_Class, Xw_Move_Focus, Xw_Panel_Widget_Class,
Xw_Popup_Mgr_Widget_Class, Xw_Popupmgr_Widget_Class,
Xw_Primitive_Widget_Class, Xw_Push_Button_Widget_Class,
Xw_Row_Col_Widget_Class, Xw_Sash_Widget_Class,
Xw_Scroll_Bar_Widget_Class, Xw_Scrollbar_Widget_Class,
Xw_Scrolled_Window_Widget_Class, Xt_Set_Arg, Xw_Sraster_Widget_Class,
Xw_Static_Raster_Widget_Class, Xw_Static_Text_Widget_Class,
Xw_Statictext_Widget_Class, Xw_String_Source_Create,
Xw_String_Source_Destroy, Xw_Swindow_Widget_Class,
Xw_Text_Clear_Buffer, Xw_Text_Copy_Buffer, Xw_Text_Copy_Selection,
Xw_Text_Edit_Widget_Class, Xw_Textedit_Widget_Class,
Xw_Text_Get_Insert_Pos, Xw_Text_Get_Last_Pos, Xw_Text_Get_Selection_Pos,
Xw_Text_Insert, Xw_Text_Read_Sub_String, Xw_Text_Redraw,
Xw_Text_Set_Insert_Pos, Xw_Text_Set_Selection, Xw_Text_Set_Source,
Xw_Text_Replace, Xw_Text_Unset_Selection,
Xw_Text_Update, Xw_Text_Set_Source,
Xw_Title_Bar_Widget_Class, Xw_Titlebar_Widget_Class,
Xw_Toggle_Widget_Class, Xw_Valuator_Widget_Class,
Xw_Work_Space_Widget_Class

Intrinsics: Null_Caddr_T, Null_Widget, Null_Widget_Class

Key Bindings: Do_Goto_End_Of_History, Do_Goto_Start_Of_History,
Do_Goto_Beg_Of_Line, Do_Goto_End_Of_Line,
Do_Move_Left, Do_Move_Right,
Do_Show_History, Do_Delete, Do_Delete_This_Char,
Do_Self_Insert, Do_Kill_Line,

Do_Insert_String, Do_Show_History_Limit, Do_Set_History_Limit,
Do_Quoted_Insert, Do_Refresh_Current_Line,
Do_Refresh_Current_Line_And_Prompt, Do_Rewrite_Current_Line,
Do_Rewrite_Current_Line_And_Prompt, Do_Get_Current_Line,
Do_Get_Current_Character, Do_Get_Current_Column
Make_Binding, Interpret_String, History_Back, History_Forward

Manipulate Scope: Delete, Deuse, Undelete

Object Lister: List

Program Objects: Compile, Edit_And_Interpret

Program Units: Close, Create, Delete, Deuse, Form, Is_Open, List, Mode, Name,
Open, Put

Stars Tools: Check_Style, Count_Features, Count_Statements,
Diana_Browser, Diana_Front_End, Diana_Mklib, Diana_Rmlib,
Diana_Cleanlib, Diana_Make_Predefined_Env, Diana_Create_Predefined_Env,
Measure_Mccabe_Complexity, Set_Up, Test_Case_Generator,
Test_Results_Analyzer, Test_Procedures_Generator, Test_Comparator,
Test_Updater

Strings: Length, Slice

Text Io:

File Management: Close, Create, Delete, Form, Is_Open, Mode, Name,
Open, Reset

Default Io Control: Current_Input, Current_Output, Set_Input, Set_Output,
Standard_Input, Standard_Output

Specify Line and Page Length: Line_Length, Page_Length, Set_Line_Length,
Set_Page_Length

Column, Line and Page Control: Col, End_Of_File, End_Of_Line, End_Of_Page,
Line, New_Line, New_Page, Page, Set_Col, Set_Line, Skip_Line, Skip_Page

Character Input-Output: Get, Put

String Input-Output: Get, Get_Line, Put, Put_Line

Ace Integer Io: Get, Put

Text Objects: Edit, Edit_File, Get_File, List, List_File, Print, Put_File

Widgets Package: Action_Procedure_Pointer, Callback_Procedure_Pointer, Create, Get, Make_Xt_String, Null_Xrm_Option_List, Null_Xt_Arg_List, Put, Xt_Add_Actions, Xt_Add_Callback, Xt_App_Next_Event, Xt_Augment_Translations, Xt_Create_Managed_Widget, Xt_Create_Widget, Xt_Default_App_Context, Xt_Destroy_Widget, Xt_Dispatch_Event, Xt_Get_Value, Xt_Initialize, Xt_Main_Loop, Xt_Override_Translations, Xt_Parse_Translation_Table, Xt_Realize_Widget, Xt_Set_Arg, Xt_Set_Values, X_Text_Width

Window Draw Routines: Draw_Dashed_Line, Draw_Line, Draw_Rectangle, Draw_Rectangle_Builtin, Draw_Text

X Windows: Ascent, Descent, Text_Width

7 Abstract Data Types of ACE

This section provides a description of the Abstract Data Types (ADTs) currently supported by ACE. The ADTs are grouped into several related areas as ACE library files. They are: *standard.ace*, *commands.ace*, *bindings.ace*, *windows.ace*, and *cais.ace*.

1. *standard.ace* provides the standard Ada package, and several packages that are standard to ACE.
2. *commands.ace* provides the operations that are normally associated with a command language.
3. *bindings.ace* provides the operation to bind arbitrary key strokes to editing and history functions.
4. *windows.ace* provides a simple interface to the X Window System.
5. *cais.ace* provides an interface to the underlying CAIS-A implementation.
6. *cpu.time.ace* provides the operation to measure CPU time use.

7.1 Standard Packages

The following packages provide the standard definitions of the required Ada packages, and several packages that are standard for ACE.

7.1.1 Objects

The basic building blocks of all ACE objects are the objects package, derived types, and derived subprograms. The "Object_Type" type is the basic representation of every object in ACE, and provides the means (through derived subprograms) of defining operations on all ACE objects. Since every object in ACE is a derived type of "Object_Type" in the objects package, it is possible to define an operation that can act on every object in ACE.

package Objects is

 type Object_Type is new Integer;

end Objects;

7.1.2 Standard

This is the package "Standard" from the Ada reference manual as implemented in ACE. The description order of this package is slightly rearranged from that within the Ada reference manual order to conform with the format used in the description of all ACE's ADTs.

package Standard is

 use Objects;

 -- type Boolean is (False, True);

 -- The predefined relational operators for this type are as follows:

 function "=" (Left, Right : Boolean) return Boolean;

 function "/=" (Left, Right : Boolean) return Boolean;

 function "<" (Left, Right : Boolean) return Boolean;

 function "<=" (Left, Right : Boolean) return Boolean;

 function ">" (Left, Right : Boolean) return Boolean;

 function ">=" (Left, Right : Boolean) return Boolean;

 -- The predefined logical operators and the predefined logical

 -- negation operator are as follows:

 function "and" (Left, Right : Boolean) return Boolean;

 function "or" (Left, Right : Boolean) return Boolean;

 function "xor" (Left, Right : Boolean) return Boolean;

```
function "not" (Right : Boolean) return Boolean;

-- the Universal type universal_integer is predefined.

-- type Integer is range -Integer'last .. Integer'last;

-- The predefined operators for this type are as follows:

function "=" (Left, Right : Integer) return Boolean;
function "/=" (Left, Right : Integer) return Boolean;
function "<" (Left, Right : Integer) return Boolean;
function "<=" (Left, Right : Integer) return Boolean;
function ">" (Left, Right : Integer) return Boolean;
function ">=" (Left, Right : Integer) return Boolean;


function "+" (Right : Integer) return Integer;
function "-" (Right : Integer) return Integer;
function "abs" (Right : Integer) return Integer;


function "+" (Left, Right : Integer) return Integer;
function "-" (Left, Right : Integer) return Integer;
function "*" (Left, Right : Integer) return Integer;
function "/" (Left, Right : Integer) return Integer;
function "rem" (Left, Right : Integer) return Integer;
function "mod" (Left, Right : Integer) return Integer;


function "***" (Left : Integer; Right : Integer) return Integer;

-- Type Float is unimplemented.

-- Type Character is partially implemented, but not supported.

package Ascii is
```

-- Control characters:

```
Nul      : constant Character := Character'VAL (0);
Soh      : constant Character := Character'VAL (1);
Stx      : constant Character := Character'VAL (2);
Etx      : constant Character := Character'VAL (3);
Eot      : constant Character := Character'VAL (4);
Enq      : constant Character := Character'VAL (5);
Ack      : constant Character := Character'VAL (5);
Bel      : constant Character := Character'VAL (7);
Bs       : constant Character := Character'VAL (8);
Ht       : constant Character := Character'VAL (9);
Lf       : constant Character := Character'VAL (10);
Vt       : constant Character := Character'VAL (11);
Ff       : constant Character := Character'VAL (12);
Cr       : constant Character := Character'VAL (13);
So       : constant Character := Character'VAL (14);
Si       : constant Character := Character'VAL (15);
Dle      : constant Character := Character'VAL (16);
Dc1      : constant Character := Character'VAL (17);
Dc2      : constant Character := Character'VAL (18);
Dc3      : constant Character := Character'VAL (19);
Dc4      : constant Character := Character'VAL (20);
Nak      : constant Character := Character'VAL (21);
Syn      : constant Character := Character'VAL (22);
Etb      : constant Character := Character'VAL (23);
Can      : constant Character := Character'VAL (24);
Em       : constant Character := Character'VAL (25);
Sub      : constant Character := Character'VAL (26);
Esc      : constant Character := Character'VAL (27);
Fs       : constant Character := Character'VAL (28);
Gs       : constant Character := Character'VAL (29);
Rs       : constant Character := Character'VAL (30);
Us       : constant Character := Character'VAL (31);
Del      : constant Character := Character'VAL (127);
```

-- Other characters:

```
Exclam   : constant Character := '!';
Sharp    : constant Character := '#';
Percent  : constant Character := '%';
Colon    : constant Character := ':';
Query    : constant Character := '?';
L_Bracket : constant Character := '[';
R_Bracket : constant Character := '];
```

```
Underline : constant Character := '_';
L_Brace   : constant Character := '{';
R_Brace   : constant Character := '}';

Quotation : constant Character := '"';
Dollar     : constant Character := '$';
Ampersand  : constant Character := '&';
Semicolon : constant Character := ';';
At_Sign    : constant Character := '@';
Back_Slash : constant Character := '\';
Circumflex : constant Character := '^';
Grave      : constant Character := '`';
Bar        : constant Character := '|';
Tilde     : constant Character := '~';
```

-- Lower case letters:

```
Lc_A      : constant Character := 'a';
Lc_B      : constant Character := 'b';
Lc_C      : constant Character := 'c';
Lc_D      : constant Character := 'd';
Lc_E      : constant Character := 'e';
Lc_F      : constant Character := 'f';
Lc_G      : constant Character := 'g';
Lc_H      : constant Character := 'h';
Lc_I      : constant Character := 'i';
Lc_J      : constant Character := 'j';
Lc_K      : constant Character := 'k';
Lc_L      : constant Character := 'l';
Lc_M      : constant Character := 'm';
Lc_N      : constant Character := 'n';
Lc_O      : constant Character := 'o';
Lc_P      : constant Character := 'p';
Lc_Q      : constant Character := 'q';
Lc_R      : constant Character := 'r';
Lc_S      : constant Character := 's';
Lc_T      : constant Character := 't';
Lc_U      : constant Character := 'u';
Lc_V      : constant Character := 'v';
Lc_W      : constant Character := 'w';
Lc_X      : constant Character := 'x';
Lc_Y      : constant Character := 'y';
Lc_Z      : constant Character := 'z';
```

end Ascii;

-- Predefined subtypes:

subtype Natural is Integer; -- range 0 .. Integer'last;

subtype Positive is Integer; -- range 1 .. Integer'last;

-- type String is array (Positive range <>) of Character;

-- Type "String" is not implemented in ACE as a one-dimensional array

-- of the predefined type character. Strings and string literals

-- within ACE provide some of the operations that are applicable to

-- Standard strings. Operations applicable to one-dimensional arrays

-- are not applicable to "String"s in ACE.

--

-- When composite types are supported in ACE, "String" will be changed

-- to its array definition.

subtype Ace_String is String;

function "=" (Left, Right : String) return Boolean;

function "/=" (Left, Right : String) return Boolean;

function "<" (Left, Right : String) return Boolean;

function "<=" (Left, Right : String) return Boolean;

function ">" (Left, Right : String) return Boolean;

function ">=" (Left, Right : String) return Boolean;

function "&" (Left : String; Right : String) return String;

function "&" (Left : String; Right : Character) return String;

function "&" (Left : Character; Right : String) return String;

function "&" (Left : Character; Right : Character) return String;

type Duration is new Object_Type;

-- Duration is simply a new integer type

-- since fixed points are not implemented [TDB]

-- Exceptions are not supported. [TDB]

-- Constraint_Error : exception;

-- Numeric_Error : exception;

-- Program_Error : exception;

-- Storage_Error : exception;

```
-- Tasking_Error : exception;

end Standard;
```

7.1.3 System

Predefined system package, as defined in the Ada standard.

```
package System is
  use Objects;

  type Address is new Object_Type;
  type Name is (Ms_Dos, Sun_Unix);

  System_Name : constant Name := Sun_Unix;

  -- Storage_Unit : constant Integer := 1;
  -- Memory_Size  : constant Integer := 1;

  Min_Int : constant Integer := Integer'First;
  Max_Int : constant Integer := Integer'Last;
  -- Max_Digits   : constant Integer := 1;
  -- Max_Mantissa  : constant Integer := 1;
  -- Fine_Delta   : constant Integer := 1;
  -- Tick         : constant Integer := 1;

  -- subtype PRIORITY is integer range -16 .. 16;

end System;
```

7.1.4 Io_Exceptions

This package defines the exceptions needed by the packages Sequential_Io, Direct_Io, and Text_Io. Only Text_Io is implemented in ACE. Exceptions are currently not fully supported.

```
package Io_Exceptions is

  Status_Error : exception;
  Mode_Error   : exception;
  Name_Error   : exception;
  Use_Error    : exception;
```

```
Device_Error : exception;  
End_Error    : exception;  
Data_Error   : exception;  
Layout_Error : exception;
```

```
end Io_Exceptions;
```

7.1.5 Low_Level_Io

Low Level input-output operations are operations that act on a physical device. Low_Level_Io is currently not supported.

```
package Low_Level_Io is
```

```
end Low_Level_Io;
```

7.1.6 Calendar

This package provides the user access to operations on the clock, as defined in the LRM (9.6).

Note: Duration is currently not implemented as a fixed point, and exceptions are currently not supported.

```
package Calendar is  
  use Objects;
```

```
  type Time is new Object_Type;
```

```
  subtype Year_Number is Integer; -- range 1901 .. 2099;  
  subtype Month_Number is Integer; -- range 1 .. 12;  
  subtype Day_Number is Integer; -- range 1 .. 31;  
  subtype Day_Duration is Duration; -- range 0 .. 86_400;
```

```
  function Clock return Time;
```

```
  function Year (Date : Time) return Year_Number;  
  function Month (Date : Time) return Month_Number;  
  function Day (Date : Time) return Day_Number;  
  function Seconds (Date : Time) return Day_Duration;
```

```
  procedure Split (Date      : in Time;  
                  Year       : out Year_Number;
```

```

        Month    : out Month_Number;
        Day      : out Day_Number;
        Seconds  : out Day_Duration);

function Time_Of (Year    : Year_Number;
                 Month    : Month_Number;
                 Day      : Day_Number;
                 Seconds  : Day_Duration := 0) return Time;

function "+" (Left : Time;      Right : Duration) return Time;
function "+" (Left : Duration; Right : Time)      return Time;
function "-" (Left : Time;      Right : Duration) return Time;
function "-" (Left : Time;      Right : Time)      return Duration;

function "<" (Left, Right : Time) return Boolean;
function "<=" (Left, Right : Time) return Boolean;
function ">" (Left, Right : Time) return Boolean;
function ">=" (Left, Right : Time) return Boolean;

Time_Error : exception;

```

```
end Calendar;
```

7.1.7 Text_Io

This is the "Text_Io" package of Ada (Chapter 14 of the Ada Reference Manual).

```

package Text_Io is
  use Objects;

  type File_Type is new Object_Type;
  type File_Mode is (In_File, Out_File);

  type Count is new Integer;
  subtype Positive_Count is Count;
  Unbounded : constant Count := 0; -- line and page length

  subtype Field is Integer;
  subtype Number_Base is Integer;

  type Type_Set is (Lower_Case, Upper_Case);

  -- File Management

```

```
procedure Create (File : in out File_Type;
                  Mode : in File_Mode := Out_File;
                  Name : in String    := "";
                  Form : in String    := "");

procedure Open (File : in out File_Type;
               Mode : in File_Mode;
               Name : in String;
               Form : in String := "");

procedure Close (File : in out File_Type);
procedure Delete (File : in out File_Type);
procedure Reset (File : in out File_Type;
                Mode : in File_Mode);
procedure Reset (File : in out File_Type);

function Mode (File : in File_Type) return File_Mode;
function Name (File : in File_Type) return String;
function Form (File : in File_Type) return String;

function Is_Open (File : in File_Type) return Boolean;

-- Control of default input and output files

procedure Set_Input (File : in File_Type);
procedure Set_Output (File : in File_Type);

function Standard_Input return File_Type;
function Standard_Output return File_Type;

function Current_Input return File_Type;
function Current_Output return File_Type;

-- Specification of line and page lengths

procedure Set_Line_Length (File : in File_Type;
                           To   : in Count);
procedure Set_Line_Length (To : in Count);

procedure Set_Page_Length (File : in File_Type;
                           To   : in Count);
procedure Set_Page_Length (To : in Count);
```

```
function Line_Length (File : in File_Type) return Count;
function Line_Length return Count;

function Page_Length (File : in File_Type) return Count;
function Page_Length return Count;

-- Column, Line, and Page Control

procedure New_Line (File      : in File_Type;
                   Spacing : in Positive_Count := 1);
procedure New_Line (Spacing : in Positive_Count := 1);

procedure Skip_Line (File      : in File_Type;
                   Spacing : in Positive_Count := 1);
procedure Skip_Line (Spacing : in Positive_Count := 1);

function End_Of_Line (File : in File_Type) return Boolean;
function End_Of_Line return Boolean;

procedure New_Page (File : in File_Type);
procedure New_Page;

procedure Skip_Page (File : in File_Type);
procedure Skip_Page;

function End_Of_Page (File : in File_Type) return Boolean;
function End_Of_Page return Boolean;

function End_Of_File (File : in File_Type) return Boolean;
function End_Of_File return Boolean;

procedure Set_Col (File : in File_Type;
                  To   : in Positive_Count);
procedure Set_Col (To   : in Positive_Count);

procedure Set_Line (File : in File_Type;
                  To   : in Positive_Count);
procedure Set_Line (To   : in Positive_Count);

function Col (File : in File_Type) return Positive_Count;
function Col return Positive;
```

```
function Line (File : in File_Type) return Positive_Count;
function Line return Positive_Count;
```

```
function Page (File : in File_Type) return Positive_Count;
function Page return Positive_Count;
```

```
-- Character Input-Output
```

```
procedure Get (File : in File_Type;
               Item : out Character);
procedure Get (Item : out Character);
```

```
procedure Put (File : in File_Type;
               Item : in Character);
procedure Put (Item : in Character);
```

```
-- String Input-Output
```

```
procedure Get (File : in File_Type;
               Item : out String);
procedure Get (Item : out String);
```

```
procedure Put (File : in File_Type;
               Item : in String);
procedure Put (Item : in String);
```

```
procedure Get_Line (File : in File_Type;
                    Item : out String;
                    Last : out Natural);
procedure Get_Line (Item : out String;
                    Last : out Natural);
procedure Put_Line (File : in File_Type;
                    Item : in String);
procedure Put_Line (Item : in String);
```

```
-- Instantiated generic package for Input-Output of Integer Types
```

Ace_Integer_Io is a hand-instantiated Integer_Io package.

```
package Ace_Integer_Io is
```

```
    Default_Width : Field      := 10;
    Default_Base  : Number_Base := 10;
```

```
    procedure Get (File : in File_Type;
```

```
Item : out Integer;  
Width : in Field := 0);
```

```
procedure Get (Item : out Integer;  
              Width : in Field := 0);
```

```
procedure Put (File : in File_Type;  
              Item : in Integer;  
              Width : in Field      := Default_Width;  
              Base : in Number_Base := Default_Base);  
procedure Put (Item : in Integer;  
              Width : in Field      := Default_Width;  
              Base : in Number_Base := Default_Base);
```

```
procedure Get (From : in String;  
              Item : out Integer;  
              Last : out Positive);
```

```
procedure Put (To : out String;  
              Item : in Integer;  
              Base : in Number_Base := Default_Base);
```

```
end Ace_Integer_Io;
```

```
end Text_Io;
```

7.1.8 Ace_Standard

In addition to the Ada package "Standard", ACE contains an additional set of objects and operations that are standard for ACE.

```
package Ace_Standard is
```

```
  subtype Interpreter_String is Ace_String;  
  subtype File_System_String is Ace_String;  
  subtype Data_String is Ace_String;  
  subtype Host_Os_String is Ace_String;
```

```
  type List_Mode is (Both, Specification, Implementation);
```

```
  type Method_Of_Execution is (Foreground, Background);
```

```
--  
-- These should remain in Text_Io, since File_Type  
-- should be a limited private type.  
--  
package Ace_Io is  
    use Text_Io;  
  
    procedure Set_Ace_Input (File : in File_Type);  
    procedure Set_Ace_Output (File : in File_Type);  
    function Standard_Ace_Input return File_Type;  
    function Standard_Ace_Output return File_Type;  
    function Current_Ace_Input return File_Type;  
    function Current_Ace_Output return File_Type;  
  
end Ace_Io;  
  
end Ace_Standard;
```

7.1.9 Strings

Temporary string operations that perform slices.

```
package Strings is  
  
    function Slice (Str      : in String;  
                   Start_Pos : in Integer;  
                   Stop_Pos  : in Integer) return String;  
  
    function Slice (Str : in String;  
                   Pos  : in Integer) return Character;  
  
    function Length (Str : in String) return Integer;  
  
end Strings;
```

7.2 Command Language Commands

The following packages provide operations that are similar to the expected commands provided by most command languages (i.e., *Set_Directory*).

7.2.1 Ace_Adt

Ace_Adt provides an interface to the interpret in the ACE system.

```
package Ace_Adt is
  use Text_IO;
  use Objects;
```

```
  type Ace_Statement_Database is new Object_Type;
```

This routine will "compile" a string into the statement database form.

```
  procedure Compile (Str    : in String;
                    Tree    : in out Ace_Statement_Database;
                    Error   : out Boolean);
```

This routine will "compile" a file into the statement database form.

```
  procedure Compile (File   : in File_Type;
                    Tree    : in out Ace_Statement_Database;
                    Error   : out Boolean);
```

This routine will delete the statement database tree associated with "tree".

```
  procedure Delete (Tree : in Ace_Statement_Database);
```

Interpret will "execute" the tree associated with the parameter "tree".

```
  procedure Interpret (Tree : in Ace_Statement_Database;
                     Error  : out Boolean);
```

Interpret will "execute" the value of the Str parameter.

```
  procedure Interpret (Str    : in String;
                     Error   : out Boolean);
```

Interpret will "execute" the contents of a file.

```
  procedure Interpret (File   : in File_Type;
                     Error   : out Boolean);
```

Interpret will "execute" the tree associated with the parameter "tree".

```
  procedure Interpret (Tree : in Ace_Statement_Database);
```

Interpret will "execute" the value of the Str parameter.

```
  procedure Interpret (Str    : in String);
```

Interpret will "execute" the contents of a file.

```
  procedure Interpret (File   : in File_Type);
```

```
Interpret_File will "execute" the contents of a file.  
  procedure Interpret_File (File : String);  
end Ace_Adt;
```

7.2.2 Host_Os

"Host_Os" provides the interface to the underlying operating system on which ACE is executing.

package Host_Os is

Host provides an escape to the host operating system. The text of the "Command" parameter will be passed to the command language processor of the host operating system. Arguments to the command may be passed, as well as an indication whether ACE should wait for the completion of the command.

```
  procedure Host (Command           : in Host_Os_String;  
                  Command_Arguments : in Host_Os_String := "";  
                  How                : in Method_of_Execution := Foreground);
```

```
end Host_Os;
```

7.2.3 Manipulate_Scope

"Manipulate_Scope" contains routines which allow the scope of objects to be manipulated within ACE. These routines support the dynamic removal of existing objects and the availability to "undo" the Ada "Use" statement.

package Manipulate_Scope is

Procedure "Delete" removes an object from the ACE environment. This allows objects to be removed and a new definition of the object to be reintroduced into ACE. The Form parameter also allows overloaded names to be uniquely identified.

```
  procedure Delete (Name : in Interpreter_String;  
                   Form : in Interpreter_String := "");
```

```
-- [TBD] NOT YET IMPLEMENTED
```

```
-- procedure Undelete (Name : in Interpreter_String;  
--                    Form : in Interpreter_String);
```

```
-- procedure Undelete (Name : in Interpreter_String);
```

Procedure "Deuse" performs an "undo" for the Ada "Use" statement. The names given direct visibility are no longer directly visible. NOTE: an item from another package that was hidden due to USE-ing this package will not automatically be made visible.

```
procedure Deuse (Package_Name : in Interpreter_String;
                Form           : Interpreter_String := "");
```

```
end Manipulate_Scope;
```

7.2.4 Debugger

The debugger package provides routines to symbolically view the execution of programs.

package Debugger is

This procedure will list the contents of the entire statement database (i.e., every object declared in the environment).

```
procedure List_Symbol_Table (List_Kind : in List_Mode := Specification);
```

This procedure will list the object that has been associated with the "name" parameter. The List_Kind parameter can be used to list the implementation or specification of the object, and the Form parameter can be used to identify an overloaded subprogram.

```
procedure List (Name : in Interpreter_String;
               List_Kind : in List_Mode := Specification;
               Form : in Interpreter_String := "");
```

This procedure will list the specification of the object that has been associated with the "name" parameter. The Form parameter can be used to identify an overloaded subprogram.

```
procedure List (Name : in Interpreter_String;
               Form : in Interpreter_String := "");
```

List_Breakpoints will list all the break points that are currently active.

```
procedure List_Breakpoints;
```

Break will temporarily suspend execution for debugging.
procedure Break;

Continue will begin execution after a Break statement was encountered
procedure Continue;

Step will execute a statement after a break statement was encountered
procedure Step (Count : Positive := 1);

Set_Break will place a break point on the appropriate Statement_Number
procedure Set_Break (Statement_Number : in Integer);
Set_Break will place a break point on the appropriate Subprogram
procedure Set_Break (Name : in Interpreter_String);
Set_Break will place a break point on the current Statement
procedure Set_Break;

Clear_Break will remove a break point on the appropriate Statement_Number
procedure Clear_Break (Statement_Number : in Integer);
Clear_Break will remove a break point on the appropriate Subprogram
procedure Clear_Break (Name : in Interpreter_String);
Clear_Break will remove a break point on the current statement
procedure Clear_Break;

Display will list the current statement
procedure Display (Statement_Number : in Integer);
Display will list the first statement associated with a
subprogram upon which a breakpoint may be set.
procedure Display (Name : in Interpreter_String);
Display will list the current statement.
procedure Display;

Display_Current_Statement will list the current statement.
procedure Display_Current_Statement;

Display_Next will list the next executable statement
procedure Display_Next (Statement_Number : in Integer);
Display_Next will list the next executable statement
procedure Display_Next;

Display_Previous will list the previously executed statement
procedure Display_Previous (Statement_Number : in Integer);
Display_Previous will list the previously executed statement
procedure Display_Previous;

end Debugger;

7.2.5 File_System

This package is the basic definition of the ACE file system. All other file system packages will be derived from this one (i.e., they will use this package--directly or indirectly--and derive types from the basic type "File_Object").

package File_System is

 use Objects;

 use Text_IO;

 -- File_Object is the basic representation of an Object (or File) in
 -- ACE.

 type File_Object is new File_Type;

 type Object_Mode is (In_Object, Out_Object);

 -- by every derived type of File_Object.

 -- Other commands can be declared

 -- Using this technique we can define the entire file

 -- system for ACE as Ada declarations.

Generate a temporary name.

 function Temporary_Name return String;

This procedure copies the contents from one file to the other file.

 procedure Copy (From : in File_Object;
 To : in File_Object);

This procedure copies the contents from one file to the other file.

 procedure Copy (From : in File_System_String;
 To : in File_System_String);

This procedure deletes an object from the file system.

 procedure Delete_File (Obj : in File_System_String);

This procedure changes the name associated with a file.

 procedure Rename (From : in File_Object;
 To : in File_Object);

This procedure changes the name associated with a file.

```

procedure Rename (From : in File_System_String;
                  To   : in File_System_String);

```

This function returns the boolean value "True" if a file exists within the current working directory.

```

function Exists (Obj : in File_System_String) return Boolean;

```

This procedure creates a new persistent file object with the specified name. If no name is given, an arbitrary name will be generated.

```

procedure Create (Obj  : in out File_Object;
                  Mode : in Object_Mode      := Out_Object;
                  Name  : in File_System_String := "";
                  Form  : in Host_Os_String    := "");

```

This procedure associates a file object with the persistent object having the specified name.

```

procedure Open (Obj  : in out File_Object;
               Mode : in Object_Mode;
               Name  : in File_System_String;
               Form  : in Host_Os_String := "");

```

This procedure resets the specified file object, possibly changing the mode associated with it.

```

procedure Reset (File : in out File_Object;
                Mode  : in Object_Mode);

```

This procedure resets the specified file object.

```

procedure Reset (File : in out File_Object);

```

```

end File_System;

```

7.2.6 Directory_Objects

Directory_Objects provides operations on directories

```

package Directory_Objects is
  use File_System;

```

```

  -- There needs to be a universal naming scheme for identifying
  -- directories (e.g., "...").

```

```

  type Directory_Object is new File_Object;

```

This procedure creates a new directory object with the specified name. If a name is not given, an arbitrary name will be generated

```
procedure Create (Directory : in out Directory_Object;  
                  Name       : in File_System_String;  
                  Form       : in Host_Os_String := "");
```

This procedure associates an object with the directory object

```
procedure Open (Directory : in out Directory_Object;  
               Name       : in File_System_String;  
               Form       : in Host_Os_String := "");
```

This procedure disassociates an object from the directory object

```
procedure Close (Directory : in out Directory_Object);
```

This function returns the name of the specified directory object

```
function Name (File : in Directory_Object) return String;
```

This function returns the form string of the specified directory object

```
function Form (File : in Directory_Object) return String;
```

This function returns the file status of the specified directory object

```
function Is_Open (File : in Directory_Object) return Boolean;
```

This function returns the user's home directory

```
function Home_Directory return Directory_Object;
```

This function returns the user's current working directory

```
function Current_Directory return Directory_Object;
```

This procedure lists the contents of the specified directory object

```
procedure List (Directory : in Directory_Object := Current_Directory);
```

This procedure changes the current working directory to the specified directory object, or the specified file system directory

```
procedure Set_Directory (Directory : in Directory_Object := Home_Directory);
```

This procedure changes the current working directory to the specified directory object, or the specified file system directory

```
procedure Set_Directory (Directory : in File_System_String);
```

This procedure displays the current working directory

```
procedure Put_Current_Directory;
```

```
end Directory_Objects;
```

7.2.7 Text_Objects

Text_Objects provides operations on text files.

```
package Text_Objects is
  use File_System;

  type Text_Object is new File_Object;
  type History_Mode is (New_File, Old_File);
```

This procedure displays the contents of a text file, either by specifying the text object

```
  procedure List (Text : in Text_Object);
```

This procedure displays the contents of a text file, either by specifying the file system name of the text file

```
  procedure List_File (Text : in File_System_String);
  -- This routine should be List not List_File,
  -- but until we have expanded names this will have
  -- to do.
```

This procedure invokes the system editor upon the specified text file

```
  procedure Edit (Text : in out Text_Object;
                 How   : in Method_Of_Execution := Foreground);
```

This procedure invokes the system editor upon the specified text file

```
  procedure Edit_File (Text      : in File_System_String;
                      History    : in History_Mode := Old_File;
                      How        : in Method_Of_Execution := Foreground);
  -- These routines should be Edit not Edit_File,
  -- but until we have expanded names this will have
  -- to do.
```

This procedure will print the specified file on the standard default printer

```
  procedure Print (Text : in Text_Object);
```

This procedure will print the specified file on the standard default printer

```
  procedure Print (Text : in File_System_String);
```

This procedure will put the textual Ada code that is associated with the ACE persistent object (Interpreter_String) into the specified file name.

```
  procedure Put_File (Object      : in Interpreter_String;
```

```
Form      : in Interpreter_String;  
File_Name : in File_System_String);
```

This procedure will put the textual Ada code that is associated with the ACE persistent object (Interpreter_String) into a default file that will have a file name identical to the name of the ACE object.

```
procedure Put_File (Object : in Interpreter_String;  
Form      : in Interpreter_String := "");
```

This procedure will cause ACE to read textual input from the specified file, rather than from Ace_Standard_Input (see Ace_Standard.Ace_Io package), until the end of file is reached.

```
procedure Get_File (File_Name : in File_System_String);
```

end Text_Objects;

7.2.8 Program_Objects

Program_Objects provides operations for text files that contain Ada code.

```
package Program_Objects is  
  use File_System;  
  use Text_Objects;  
  use Ace_Adt;
```

"Program_Text" represents files that contain Ada code.
type Program_Text is new Text_Object;

"Format_Text" represents files that are formatted, such as program listings.
type Format_Text is new Text_Object;

"Data_Text" can represent any ASCII file.
type Data_Text is new Text_Object;

This procedure submits the specified program text to an Ada compiler with the Form string being passed to the Ada compiler.

```
procedure Compile (Program : in Program_Text;  
Form      : in Host_Os_String := "");
```

This procedure submits the specified program text to an Ada compiler and linker, identifying the name of the main unit and the executable code file.

```

procedure Compile (Program      : in Program_Text;
                   Main_Unit    : in Data_String;
                   Host_Binary_Name : in File_System_String);

```

This procedure submits the specified file containing Ada code to an Ada compiler

```

    procedure Compile (Program : in File_System_String;
                      Form      : in Data_String := "");

```

This procedure submits the specified program text to an Ada compiler and linker, identifying the name of the main unit and executable code file.

```

    procedure Compile (Program      : in File_System_String;
                      Main_Unit    : in Data_String;
                      Host_Binary_Name : in File_System_String);

```

This procedure edits the ACE item ("Name"), where the Form parameter is used to uniquely identify overloaded names, such as subprograms, with the ACE item also being stored in the program text file denoted by "Object". The item will be reinterpreted by the ACLI.

```

    procedure Edit_And_Interpret (Name      : in Interpreter_String;
                                Form       : in Interpreter_String := "";
                                Object      : in out Program_Text);

```

This procedure edits the ACE item ("Name"), where the Form parameter may be used to uniquely identify overloaded names, such as subprograms. The item will then be reinterpreted by the ACLI.

```

    procedure Edit_And_Interpret (Name : in Interpreter_String;
                                Form  : in Interpreter_String := "");

```

end Program_Objects;

7.2.9 Program_Text_Objects

Program_Text_Objects provides operations on program specs and bodies package Program_Text_Objects is

```

    use Program_Objects;

```

"Spec_Program" represents files that contain Ada specifications.

```

    type Spec_Program is new Program_Text;

```

"Body_Program" represents files that contain Ada implementations.

```

    type Body_Program is new Program_Text;

```

end Program_Text_Objects;

7.2.10 Binary_Objects

Binary_Objects provides operations on executable files.

```
package Binary_Objects is
  use File_System;
```

Binary_Object represents files that would contain any data such as program executables, raster images, etc.

```
  type Binary_Object is new File_Object;
```

Program_Binary is the compiled version of a Program_Text type.

```
  type Program_Binary is new Binary_Object;
```

Format_Binary is formatted output that contains non-ASCII data, such as raster images.

```
  type Format_Binary is new Binary_Object;
```

Data_Binary is any file that contains non-ASCII data and is neither a Program_Binary or Format_Binary file.

```
  type Data_Binary is new Binary_Object;
```

This procedure has the host operating system execute the specified program binary object.

```
  procedure Execute (Program           : in Program_Binary;
                    Command_Line_Arguments : in Data_String := "");
  procedure Execute (Program           : in File_System_String;
                    Command_Line_Arguments : in Data_String := "");
```

```
end Binary_Objects;
```

7.2.11 Program_Units

Package "Program_Units" contains the definition of ACE compilation units. Packages and subprograms are currently supported within ACE compilations units. This package provides the definition of the creation, deletion, open, close, mode, and is_open operations.

```
package Program_Units is
  use Objects;
```

```
  type Package_Type is new Object_Type;
  type Subprogram_Type is new Object_Type;
  type Program_Unit_Mode is (In_Prog_Unit, Out_Prog_Unit);
```

```
procedure Create (Ace_Package : in out Package_Type;  
                  Mode         : in Program_Unit_Mode := Out_Prog_Unit;  
                  Name         : in Interpreter_String := "";  
                  Form         : in Interpreter_String := "");
```

```
--procedure Create (Ace_Subprogram : in out Subprogram_Type;  
--                  Mode         : in Program_Unit_Mode := Out_Prog_Unit;  
--                  Name         : in Interpreter_String := "";  
--                  Form         : in Interpreter_String := "");
```

Procedure "Open" associates an existing package with a package object

```
procedure Open (Ace_Package : in out Package_Type;  
               Mode         : in Program_Unit_Mode;  
               Name         : in Interpreter_String;  
               Form         : in Interpreter_String := "");
```

Procedure "Open" associates an existing program with a subprogram object

```
procedure Open (Ace_Subprogram : in out Subprogram_Type;  
               Mode         : in Program_Unit_Mode;  
               Name         : in Interpreter_String;  
               Form         : in Interpreter_String := "");
```

This procedure disassociates a package object with a package resident within ACE

```
procedure Close (Ace_Package : in out Package_Type);
```

This procedure disassociates a subprogram object with a package resident within ACE

```
procedure Close (Ace_Subprogram : in out Subprogram_Type);
```

This procedure removes the specified package object from the name space of ACE.

```
procedure Delete (Ace_Package : in out Package_Type);
```

This procedure removes the specified subprogram object from the name space of ACE.

```
procedure Delete (Ace_Subprogram : in out Subprogram_Type);
```

This function returns the mode of the specified package object

```
function Mode (Ace_Package : in Package_Type) return Program_Unit_Mode;
```

This function returns the mode of the specified subprogram object

```
function Mode (Ace_Subprogram : in Subprogram_Type)
    return Program_Unit_Mode;
```

This function returns the name of the specified package object

```
function Name (Ace_Package : in Package_Type) return Interpreter_String;
```

This function returns the name of the specified subprogram object

```
function Name
    (Ace_Subprogram : in Subprogram_Type) return Interpreter_String;
```

This function returns the form of the specified package object.
The form is the statement number associated with the package.

```
function Form (Ace_Package : in Package_Type) return Interpreter_String;
```

This function returns the form of the specified subprogram object.
The form is the statement number associated with the SubProgram.

```
function Form
    (Ace_Subprogram : in Subprogram_Type) return Interpreter_String;
```

This function returns the open status of the specified package object

```
function Is_Open (Ace_Package : in Package_Type) return Boolean;
```

This function returns the open status of the specified subprogram object

```
function Is_Open (Ace_Subprogram : in Subprogram_Type) return Boolean;
```

This procedure performs an "undo" of the Ada "use" statement.
The names given direct visibility by the "use" statement will no longer be directly visible.

note: An item from another package that was hidden due to using the package will not automatically be made visible.

```
procedure Deuse (Ace_Package : in Package_Type);
```

The location type specifies whether the subprogram is only to be moved into the package specification or into the package body.

```
type Location_Type is (In_Spec, In_Body);
```

The visibility type specifies whether the the subprogram object is to be visible (moved into the visible portion of the package spec) or hidden (moved into the private portion of the package spec).

```
type Visibility_Type is (Visible, Hidden);
```

"Put" will put the textual Ada code that is associated with the ACE persistent object (Subprogram_Item) into

the specified package.

```

procedure Put (Subprogram_Item : in Subprogram_Type;
               Into_Package     : in Package_Type;
               Location         : in Location_Type;
               Visibility       : in Visibility_Type);

```

"List" displays the package object. The "List_Method" controls the portion of the package listed: spec, body, or both.

```

procedure List (Ace_Package : in Package_Type;
               List_Method : in List_Mode := Both);

```

"List" displays the subprogram object. The "List_Method" controls the portion of the subprogram listed: spec, body, or both.

```

procedure List (Ace_Subprogram : in Subprogram_Type;
               List_Method     : in List_Mode := Both);

```

end Program_Units;

7.2.12 Help_Adt

The Help_Adt provides operations to aid the user. Type "Help;" to learn more about the help subprogram.

package Help_Adt is

The Help command provides the user with on-line assistance for declared objects in ACE: packages, subprograms, and types. These objects may be associated with help comments, which are denoted by the PDL-like "--?". When invoked, this command displays the text of the help comment associated with the object (NAME). The form parameter is used to uniquely identify overloaded names.

```

procedure Help (Name : Interpreter_String := "Help";
               Form : Interpreter_String := "");

```

end Help_Adt;

7.2.13 Object_Lister

Object_Lister provides operations that allow users to browse the defined constructs in ACE.

package Object_Lister is

```
type Object_Declarations is
  (Any,
   Objs,
   Type_Marks,
   Types,
   Subtypes,
   Subprograms,
   Packages,
   Procedures,
   Functions,
   Labels);
```

procedure List will display the names of environment objects that are currently visible. One can focus the report on a particular object

```
procedure List (Object_Type : Object_Declarations;
                Specifier    : Interpreter_String := "");
```

```
end Object_Lister;
```

7.3 Key Bindings

The following packages provide operations to bind arbitrary keystrokes to editing and history functions.

7.3.1 Key_Bindings

Key_Bindings provides operations to bind arbitrary keystrokes to editing and history functions.

```
package Key_Bindings is
```

```
type Commands is
  (Error,
   Unbound,
   Line_Separator,
   Interpret_String,
   History_Back,
   History_Forward,
   Move_Left,
   Move_Right,
   Beginning_Of_Line,
   End_Of_Line,
   Start_Of_History,
```

```

End_Of_History,
Delete,
Delete_This_Char,
Show_History,
Set_History_Limit,
Show_History_Limit,
Kill_Line,
Quoted_Insert,
Refresh_Current_Line,
Refresh_Current_Line_and_Prompt,
Rewrite_Current_Line,
Rewrite_Current_Line_and_Prompt,
Get_Current_Line,
Get_Current_Character,
Get_Current_Column,
Insert_String,
Self_Insert);

```

Bind an input string to a history or editing command;
available commands are:

Unbound	(used to remove an existing binding)
Line_Separator	End-of-line symbol for input
Interpret_String	Interpret arbitrary Ada code
History_Back	Show previous command
History_Forward	Show next command
Move_Left	Move cursor left (backspace)
Move_Right	Move cursor right
Beginning_Of_Line	Move cursor to beginning of line
Start_Of_History	Show oldest command
End_Of_History	Show most recent command
Show_History	Show complete command history
Delete	Delete character to left of cursor
Delete_This_Char	Delete character under cursor
Self_Insert	Insert character corresponding to keystroke
Show_History_Limit	Show the number of lines being saved in history
Set_History_Limit	Change the number of lines being saved in history
Kill_Line	Delete the entire line
Quoted_Insert	For the next character received, do not look for a binding, just insert it into the line
Refresh_Current_Line	refresh the portion of the line following the prompt
Refresh_Current_Line_And_Prompt	refresh the entire line
Rewrite_Current_Line	replace the current line with the argument
Rewrite_Current_Line_And_Prompt	write the prompt, followed by the argument on the next physical line

Get_Current_Line	argument will hold the current input
Get_Current_Character	argument will hold the current character, or ascii.nul if the cursor is at the end of the line
Get_Current_Column	argument is the 1-based position of the cursor
Insert_String	insert argument into current line at current position

```

procedure Make_Binding (Char_Seq      : in String;
                        Binding        : in Commands;
                        Optional_String : in String := "");

```

Bind an input character to a history or editing command;
available commands are:

Unbound	(used to remove an existing binding)
Line_Separator	End-of-line symbol for input
Interpret_String	Interpret arbitrary Ada code
History_Back	Show previous command
History_Forward	Show next command
Move_Left	Move cursor left (backspace)
Move_Right	Move cursor right
Beginning_Of_Line	Move cursor to beginning of line
Start_Of_History	Show oldest command
End_Of_History	Show most recent command
Show_History	Show complete command history
Delete	Delete character to left of cursor
Delete_This_Char	Delete character under cursor
Self_Insert	Insert character corresponding to keystroke
Show_History_Limit	Show the number of lines being saved in history
Set_History_Limit	Change the number of lines being saved in history
Kill_Line	Delete the entire line
Quoted_Insert	For the next character received, do not look for a binding, just insert it into the line
Refresh_Current_Line	refresh the portion of the line following the prompt
Refresh_Current_Line_And_Prompt	refresh the entire line
Rewrite_Current_Line	replace the current line with the argument
Rewrite_Current_Line_And_Prompt	write the prompt, followed by the argument on the next physical line
Get_Current_Line	argument will hold the current input
Get_Current_Character	argument will hold the current character, or ascii.nul if the cursor is at the end of the line
Get_Current_Column	argument is the 1-based position of the cursor
Insert_String	insert argument into current line at current position

```

procedure Make_Binding (C                : in Character;

```

```
Binding      : in Commands;  
Optional_String : in String := "";
```

-- Commands

String is the name of a procedure to be invoked
 procedure Interpret_String (Str : String);

Display previous command, that is, the next oldest command
from the current history position; if this position is the
top of the command history, this procedure does nothing
 procedure History_Back;

Display next command, that is, the next most recent command
from the current history position; if this position is the
bottom of the command history, this procedure does nothing
 procedure History_Forward;

Move cursor left (i.e., backspace); if the cursor is already
at the beginning of the input line, this procedure does nothing
 procedure Do_Move_Left;

Move cursor right; this procedure will only move the cursor
as far as one position to the right of the last input character
 procedure Do_Move_Right;

Move cursor to beginning of (input) line
 procedure Do_Goto_Beg_Of_Line;

Move the cursor to the end of the line;
 procedure Do_Goto_End_Of_Line;

Show oldest command in the command history
 procedure Do_Goto_Start_Of_History;

Show most recent command in the command history
 procedure Do_Goto_End_Of_History;

Show complete command history
 procedure Do_Show_History;

Delete character to left of cursor
procedure Do_Delete;

Delete character under cursor
procedure Do_Delete_This_Char;

Insert character corresponding to keystroke;
procedure Do_Self_Insert;

Delete the entire line
procedure Do_Kill_Line;

Insert a string passed in Str;
procedure Do_Insert_String (Str : String);

Show the user the number of lines saved for the command history
procedure Do_Show_History_Limit;

Reset the number of lines of command history to be saved (based on user input)
procedure Do_Set_History_Limit;

Insert the next character typed into the current line as is
procedure Do_Quoted_Insert;

clear, then rewrite the current line, not including the prompt
procedure Do_Refresh_Current_Line;

(cursor should already be positioned at the beginning of a line) writes
out the prompt and current line;
procedure Do_Refresh_Current_Line_And_Prompt;

kill the current line and replace it with Line, not including the prompt;
procedure Do_Rewrite_Current_Line(Line : String);

(cursor should already be positioned at the beginning of a line) kills the
current line and replaces it with Line, writing out both the prompt and Line
procedure Do_Rewrite_Current_Line_And_Prompt (Line : String);

returns the current line of input
procedure Do_Get_Current_Line (Line : out String; Length : out Natural);

returns the character on which the cursor is positioned
function Do_Get_Current_Character return Character;

```

returns the 1-based number of the column the cursor is in
    function Do_Get_Current_Column return Natural;

end Key_Bindings;

```

7.4 Windowing Commands

The following packages provide a simplified interface to a small subset of the X Window System.

7.4.1 Window_Objects

This package defines the objects that are associated with the Windows abstract data type.

```

package Window_Objects is

    use Objects;

    type Window_Type is new Object_Type;

    type Coordinate is new Integer;
    type Pixels      is new Integer;

end Window_Objects;

```

7.4.2 Ace_X_Window_System

Package "Ace_X_Window_System" defines the ACE abstraction to the X window system. This package is intended for experienced windowing system programmers. X Window system version 11, release 2 is the supported version.

```

package Ace_X_Window_System is

```

```

    use Window_Objects;

```

This procedure creates a window at the specified position with the given size and associates the window with a window object.

```

    procedure Create_Window
        (Window           : in out Window_Type;
         Horizontal_Position : in Coordinate;

```

```
Vertical_Position    : in Coordinate;  
Horizontal_Size      : in Pixels;  
Vertical_Size        : in Pixels);
```

This procedure destroys the window associated with the window object.

```
procedure Destroy_Window (Window : in out Window_Type);
```

This procedure clears the window object by painting the window with the window's background color.

```
procedure Clear_Window (Window : in out Window_Type);
```

```
end Ace_X_Window_System;
```

7.4.3 Ace_User_X_Window_System

This package defines a simplified ACE windows system abstraction to the X window system. The operation nomenclature within this package adheres to Ada paradigms, rather than to the X window system paradigms. This package is intended for novices to windowing systems and novices to the X window system. X Window System version 11, release 3 is the supported version.

```
package Ace_User_X_Window_System is
```

```
    use Window_Objects;  
    use Ace_X_Window_System;
```

This procedure creates a window at the specified position with the given size and associates the window with a window object.

```
procedure Create (Window           : in out Window_Type;  
                 Horizontal_Position : in Coordinate;  
                 Vertical_Position  : in Coordinate;  
                 Horizontal_Size     : in Pixels;  
                 Vertical_Size       : in Pixels);
```

This procedure destroys the window associated with the window object.

```
procedure Delete (Window : in out Window_Type);
```

This procedure clears the window object by painting the window with the window's background color.

```
procedure Reset (Window : in out Window_Type);
```

```
end Ace_User_X_Window_System;
```

7.4.4 Window_Draw_Routines

"Window_Draw_Routines" provides a simple set of drawing operations that may be performed in window objects. The routines rely upon the X window system and therefore should only be use when X is running.

package Window_Draw_Routines is

```
use Window_Objects;
use Ace_X_Window_System;
```

"Draw_Line" draws a line within the window object from the coordinates (x1, y1) to (x2, y2)

```
procedure Draw_Line (Window          : in Window_Type;
                     X1, Y1, X2, Y2 : in Coordinate);
```

"Draw_Dashed_Line" draws a dashed line within the window object from the coordinates (x1, y1) to (x2, y2)

```
procedure Draw_Dashed_Line (Window      : in Window_Type;
                            X1, Y1, X2, Y2 : in Coordinate);
```

"Draw_Text" draws the text string within the current window object left justified from the coordinate (x, y)

```
procedure Draw_Text (Window : in Window_Type;
                    X, Y    : in Coordinate;
                    Text    : in String);
```

"Draw_Rectangle" draws a rectangle within the window object with the upper left corner positioned at coordinate (X, Y) and of the specified width and height in pixels

```
procedure Draw_Rectangle (Window      : in Window_Type;
                          X, Y         : in Coordinate;
                          Width, Height : in Pixels);
```

"Draw_Rectangle_Builtin" draws a rectangle within the window object with the upper left corner positioned at coordinate (X, Y) and of the specified width and height in pixels

```
procedure Draw_Rectangle_Builtin
(Window      : in Window_Type;
 X, Y       : in Coordinate;
 Width, Height : in Pixels);
```

end Window_Draw_Routines;

7.5 CAIS-A Commands

The following packages provide ACE's interface to the underlying CAIS-A system.

7.5.1 CAIS_Routines

Package CAIS_Routines provides operations to view and move around the CAIS-A node model, to invoke and spawn CAIS-A processes, and to build string representations of CAIS-A argument lists.

package Cais_Routines is

```
type Cais_Node_Kinds is
  (Structural_Nodes_Only,
   File_Nodes_Only,
   Process_Nodes_Only,
   All_But_Structural_Nodes,
   All_But_File_Nodes,
   All_But_Process_Nodes,
   All_Cais_Node_Kinds);
```

```
type Cais_Relationship_Kinds is
  (Primary_Relationships,
   Secondary_Relationships,
   Both_Relationship_Kinds);
```

```
subtype Cais_Arg_List_Str_Rep is Ace_String;
```

Procedure spawn_process will use the CAIS to create a process node in the CAIS node model and start the process running in the background.

```
procedure Spawn_Process (Node_Path  : in Host_Os_String;
                        Parameters : in Host_Os_String := "");
```

Procedure invoke_process will use the CAIS to create a process node in the CAIS node model and execute the process, returning results in the Results parameter.

```
procedure Invoke_Process (Node_Path  : in Host_Os_String;
                        Parameters : in Host_Os_String := "";
                        Results    : out Host_Os_String);
```

Procedure set_current_CAIS_node will change the CURRENT_NODE in

the CAIS node model to the node whose path name is passed in.

```
procedure Set_Current_Cais_Node (Node_Path : in File_System_String);
```

Procedure current_CAIS_node returns a string representation of the primary pathname of the CURRENT_NODE in the CAIS-A node model.

```
procedure Put_Current_Cais_Node;
```

Procedure put_CAIS_node_relationships displays some or all of the relationships emanating from the CURRENT_NODE in the CAIS node model

```
procedure Put_Cais_Node_Relationships
  (Relation   : in Host_Os_String      := "*";
   Key        : in Host_Os_String      := "*";
   Rel_Kinds  : in Cais_Relationship_Kinds := Both_Relationship_Kinds;
   Node_Kinds : in Cais_Node_Kinds      := All_But_Process_Nodes);
```

Procedure put_CAIS_file_node_host_name displays the host operating system's file name for the CAIS-A file node found at node model pathname passed in. This file name is the file name of the file within the CONTENTS directory of the CAIS-A database.

```
procedure Put_Cais_File_Node_Host_Name
  (Node_Path : in File_System_String := "'CURRENT_NODE");
```

Function Create_Cais_Argument_List builds a string representation of a CAIS argument list using the string item passed in as the sole item in the returned CAIS argument list string representation.

```
function Create_Cais_Argument_List (Str : in Ace_String) return
  Cais_Arg_List_Str_Rep;
```

Function Append_To_CAIS_Arg_List appends the string item passed in to the CAIS argument list string representation also passed, and returns the resulting string representation of a CAIS list.

```
function Append_To_Cais_Arg_List (List : in Cais_Arg_List_Str_Rep;
                                   Str  : in Ace_String) return
  Cais_Arg_List_Str_Rep;
```

Function Prefix_To_CAIS_Arg_List prefixes the string item passed in to the CAIS argument list string representation also passed, and returns the resulting string representation of a CAIS list.

```
function Prefix_To_Cais_Arg_List (List : in Cais_Arg_List_Str_Rep;
                                   Str  : in Ace_String) return
  Cais_Arg_List_Str_Rep;
```

```
end Cais_Routines;
```

7.5.2 STARS_Tools

Package STARS_Tools contains the interfaces to the Unisys Q task tools integrated with the ACE/CAIS-A Baseline SEE. These include the Q13 Ada source code metrics tools, the Q14 Diana IDL tools, and the Q10 test tool suite.

package Stars_Tools is

Procedure Check_Style invokes the modified Q13 Style Checker. The tool takes its input and output files as parameters. If no input file is provided, the user is prompted for its name. Output defaults to a file named "Style_Report."

```

    procedure Check_Style (File_To_Check : in File_System_String := "";
                           Flaws_File    : in File_System_String :=
                               "Flaws_Report";
                           Style_File    : in File_System_String :=
                               "Style_Report";
                           How            : in Method_of_Execution :=
                               Foreground);

```

Procedure Count_Features invokes the Q13 feature counting tool. The tool takes its input and output files as parameters. If no input file is provided, the user is prompted for its name. Output defaults to a file named "Features_Report."

```

    procedure Count_Features (File_To_Count : in File_System_String := "";
                              Results_File  : in File_System_String :=
                                  "Features_Report";
                              How            : in Method_of_Execution :=
                                  Foreground);

```

Procedure Count_Statements invokes the Q13 statement counting tool. The tool takes its input and output files as parameters. If no input file is provided, the user is prompted for its name. Output defaults to a file named "Statements_Report."

```

    procedure Count_Statements (File_To_Count : in File_System_String := "";
                                Results_File  : in File_System_String :=
                                    "Statements_Report";
                                How            : in Method_of_Execution :=
                                    Foreground);

```

Procedure Measure_Mccabe_Complexity invokes the Q13 tool which calculates the McCabe Complexity for a source file. The tool takes its input and output files as parameters.

If no input file is provided, the user is prompted for its name.
Output defaults to a file named "McCabe_Report."

```

procedure Measure_McCabe_Complexity
  (File_To_Measure : in File_System_String := "";
   Results_File    : in File_System_String := "McCabe_Report";
   How              : in Method_of_Execution := Foreground);

```

Procedure Diana_Browser invokes the Q14 Diana IDL Library Unit Browser. It takes the name of the unit to browse as an argument, and an optional "-b" which has the Browser display bodies as well as specifications.

```

procedure Diana_Browser (Unit      : in Ace_String;
                        Options : in Ace_String := "");

```

Procedure Diana_Front_End invokes the Diana IDL Front End Processor. It takes an Ada source filename to process as an argument.

```

procedure Diana_Front_End (Source : File_System_String);

```

Procedure Diana_Mklib creates a new Diana unit library.

```

procedure Diana_Mklib;

```

Procedure Diana_Rmlib removes a Diana unit library.

```

procedure Diana_Rmlib;

```

Procedure Diana_Cleanlib resets a Diana unit library.

```

procedure Diana_Cleanlib;

```

Procedure Diana_Make_Predefined_Env builds the Diana IDL predefined environment.

```

procedure Diana_Make_Predefined_Env;

```

Procedure Diana_Create_Predefined_Env builds the Diana IDL predefined environment after the IDL has been modified.

This procedure should only be run once.

```

procedure Diana_Create_Predefined_Env;

```

Procedure Test_Case_Generator invokes the Q10 test case generator.

```

procedure Test_Case_Generator;

```

Procedure Test_Results_Analyzer invokes the Q10 tool which analyzes the test case results.

```

procedure Test_Results_Analyzer;

```

Procedure Test_Procedures_Generator invokes the Q10 tool which generates testing procedures.

```
procedure Test_Procedures_Generator;
```

Procedure Test_Comparator invokes the Q10 test comparator.

```
procedure Test_Comparator;
```

Procedure Test_Updater invokes the Q10 test updater.

```
procedure Test_Updater;
```

Procedure Set_Up invokes the Q10 routine to set up the environment.

```
procedure Set_Up;
```

```
end Stars_Tools;
```

7.6 CPU Timing Package

The following packages provide routines to measure and report CPU use.

7.6.1 Cpu_Time

Cpu_Time defines operations to determine the amount of CPU time used during a specific period.

package Cpu_Time is

```
type Time is new Integer;
```

Clock returns the current CPU time

```
function Clock return Time;
```

Difference will return the amount of time between two clock points.

```
function Difference (Stop_Time, Start_Time : Time) return Time;
```

Put_Time will print the time out in a reasonable manner.

```
procedure Put_Time (A_Time : in Time);
```

```
end Cpu_Time;
```

7.7 Xt Toolkit Interface

The following packages provide an interface to a subset of the Xt Toolkit, which is a set of procedures for building applications of the X Window System.

7.7.1 X_Windows

with Objects;

Package X_Windows provides declarations of the basic X library data types needed to use the ACE interface to the Xt toolkit. The X windows library procedures supported by ACE will also be found here.

Package X_Windows is

use Objects;

type Drawable is new Object_Type;

type Context is new Object_Type;

type Visual is new Object_Type;

type Screen is new Object_Type;

type Display is new Object_Type;

subtype Window is Drawable;

subtype Pixmap is Drawable;

type Pixels is new Integer;

subtype Coordinate is Pixels;

type Caddr_T is new System.Address;

type String_Pointer is new Object_Type;

type String_Pointer_Array is new Object_Type;

type String_List is new Object_Type;

-- type String_Pointer is access String;

-- type String_Pointer_Array is array (Natural range <>) of String_Pointer;

-- type String_List is access String_Pointer_Array;

package Fonts is

type Font is new Object_Type;

type Font_Direction is (Left_To_Right, Right_To_Left);

type Font_Record is new Object_Type;

Returns the Ascent field of its parameter.

function Ascent (F : in Font_Record) return Pixels;

pragma Builtin (Ascent, 2101);

Returns the Descent field of its parameter.

```
function Descent (F : in Font_Record) return Pixels;
pragma Builtin (Descent, 2102);
```

Returns the width of its Text parameter, in the font specified by the Font_Info parameter.

```
function Text_Width (Font_Info : in Font_Record;
                    Text       : in String) return Pixels;
pragma Builtin (Text_Width, 2103);
end Fonts;
```

```
type Gravity_Type is
  (Forget_Gravity,
   Northwest_Gravity,
   North_Gravity,
   Northeast_Gravity,
   West_Gravity,
   Center_Gravity,
   East_Gravity,
   Southwest_Gravity,
   South_Gravity,
   Southeast_Gravity,
   Static_Gravity);
```

```
package Events is
  type Event is new Object_Type;
end Events;
```

```
package Resource_Manager is
  type Xrm_Option_Desc_List is new Object_Type;
end Resource_Manager;
end X_Windows;
```

7.7.2 Renamed_Xlib_Types

with X_Windows;

Package Renamed_Xlib_Types defines the connection between some type names used by Xt routines and the equivalent type names in the basic X library.

```
package Renamed_Xlib_Types is
```

```
subtype Pixel is X_Windows.Pixels;
subtype Xt_String is X_Windows.String_Pointer;
subtype Xt_String_List is X_Windows.String_Pointer_Array;
subtype Xt_String_List_Ptr is X_Windows.String_List;
subtype Argv_Type is X_Windows.String_List;

end Renamed_Xlib_Types;
```

7.7.3 Intrinsic

```
with Objects;
with System;
with Renamed_Xlib_Types;
with X_Windows;
```

The package Intrinsic contains the type declarations common to all Xt toolkit routines.

```
package Intrinsic is
  use Objects;
```

```
type Cardinal is new Integer;
type Short_Cardinal is new Integer;

-- type Cardinal is range 0 .. (2 ** 31 - 1);
-- type Short_Cardinal is range 0 .. (2 ** 16 - 1);
```

```
subtype Position is X_Windows.Pixels;
subtype Dimension is Short_Cardinal;
```

```
type Widget is new System.Address;
type Widget_Class is new System.Address;
```

```
function Null_Widget return Widget;
function Null_Widget_Class return Widget_Class;
```

```
function Null_Caddr_T return X_Windows.Caddr_T;
```

```
type Xt_App_Context is new System.Address;
```

```
package Xt_Ancillary_Types is
```

```
  subtype Xt_Arg_Val is X_Windows.Caddr_T;
  type Xt_Arg is new Object_Type;
```

```

-- type Xt_Arg is
--   record
--     Name   : Renamed_Xlib_Types.Xt_String;
--     Value  : Xt_Arg_Val;
--   end record;

end Xt_Ancillary_Types;

end Intrinsics;

```

7.7.4 Widget_Package

```

with Objects;
with System;
with Intrinsics;
with X_Windows;
with Renamed_Xlib_Types;

```

This package offers a sample selection of Xt toolkit procedures. For detailed documentation on these routines, the user should consult Xt documentation. The names of the toolkit procedures, as given in the Xt documentation are the same as the names used here, but with the underscores removed.

```

package Widget_Package is
  use Objects;

```

```

  function Null_Xrm_Option_List
    return X_Windows.Resource_Manager.Xrm_Option_Desc_List;
  pragma Builtin (Null_Xrm_Option_List, 2201);

```

```

-- type Xt_Arg_List is
--   array (Integer range <>) of Intrinsics.Xt_Ancillary_Types.Xt_Arg;

```

```

type Xt_Arg_List is new Object_Type;

```

```

function Null_Xt_Arg_List return Xt_Arg_List;
pragma Builtin (Null_Xt_Arg_List, 2202);

```

```

type Xt_Translations is new System.Address;

```

Converts an Ada string to a string suitable for the Xt toolkit.

```

function Make_Xt_String (S : in String) return Renamed_Xlib_Types.Xt_String;
pragma Builtin (Make_Xt_String, 2203);

```

Ada version of the Xt toolkit routine XtInitialize.

```

procedure Xt_Initialize
  (Out_Value : out Intrinsics.Widget;
   Name      : in String;
   Classname : in String;
   Urlist    : in X_Windows.Resource_Manager.Xrm_Option_Desc_List;
   Num_Urs   : in Intrinsics.Cardinal;
   Argc      : in out Intrinsics.Cardinal;
   Argv      : in out Renamed_Xlib_Types.Argv_Type);
pragma Builtin (Xt_Initialize, 2204);

```

This set of overloaded procedures provides a strongly typed,

Ada version of the Xt toolkit macro XtSetArg.

Current overloading permits the Value parameter to be one of the following types:

Boolean

String

Integer

Address

Dimension

Position

Gravity_Type

```

procedure Xt_Set_Arg (Arg   : out Intrinsics.Xt_Ancillary_Types.Xt_Arg;
                     Name  : in String;
                     Value : in Boolean);
pragma Builtin (Xt_Set_Arg, 2205);

```

This set of overloaded procedures provides a strongly typed,

Ada version of the Xt toolkit macro XtSetArg.

Current overloading permits the Value parameter to be one of the following types:

Boolean

String

Integer

Address

Dimension

Position

Gravity_Type

```

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                     Subscript : in Integer;
                     Name      : in String;
                     Value     : in Boolean);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg. Current overloading permits the Value parameter to be one of the following types:

```
Boolean
String
Integer
Address
Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (Arg   : out Intrinsic.Xt_Ancillary_Types.Xt_Arg;
                      Name  : in String;
                      Value : in String);
pragma Builtin (Xt_Set_Arg, 2206);
```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg. Current overloading permits the Value parameter to be one of the following types:

```
Boolean
String
Integer
Address
Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value      : in String);
```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg. Current overloading permits the Value parameter to be one of the following types:

```
Boolean
String
Integer
Address
Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (Arg   : out Intrinsic.Xt_Ancillary_Types.Xt_Arg;
```

```

        Name : in String;
        Value : in Integer);
pragma Builtin (Xt_Set_Arg, 2207);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg. Current overloading permits the Value parameter to be one of the following types:

```

    Boolean
    String
    Integer
    Address
    Dimension
    Position
    Gravity_Type
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                     Subscript : in Integer;
                     Name      : in String;
                     Value     : in Integer);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg. Current overloading permits the Value parameter to be one of the following types:

```

    Boolean
    String
    Integer
    Address
    Dimension
    Position
    Gravity_Type
procedure Xt_Set_Arg (Arg      : out Intrinsics.Xt_Ancillary_Types.Xt_Arg;
                     Name     : in String;
                     Value    : in System.Address);
pragma Builtin (Xt_Set_Arg, 2208);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg. Current overloading permits the Value parameter to be one of the following types:

```

    Boolean
    String
    Integer
    Address

```

```

Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in System.Address);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg.

Current overloading permits the Value parameter to be one of the following types:

```

Boolean
String
Integer
Address
Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (Arg      : out Intrinsic.Xt_Ancillary_Types.Xt_Arg;
                      Name     : in String;
                      Value    : in Intrinsic.Dimension);
pragma Builtin (Xt_Set_Arg, 2209),

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg.

Current overloading permits the Value parameter to be one of the following types:

```

Boolean
String
Integer
Address
Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Intrinsic.Dimension);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg.

Current overloading permits the Value parameter to be one of the following types:

```

Boolean
String
Integer
Address
Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (Arg   : out Intrinsic.Xt_Ancillary_Types.Xt_Arg;
                      Name   : in String;
                      Value   : in Intrinsic.Position);
pragma Builtin (Xt_Set_Arg, 2210);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg. Current overloading permits the Value parameter to be one of the following types:

```

Boolean
String
Integer
Address
Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Intrinsic.Position);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg. Current overloading permits the Value parameter to be one of the following types:

```

Boolean
String
Integer
Address
Dimension
Position
Gravity_Type
procedure Xt_Set_Arg (Arg   : out Intrinsic.Xt_Ancillary_Types.Xt_Arg;
                      Name   : in String;
                      Value   : in X_Windows.Gravity_Type);
pragma Builtin (Xt_Set_Arg, 2211);

```

This set of overloaded procedures provides a strongly typed, Ada version of the Xt toolkit macro XtSetArg.

Current overloading permits the Value parameter to be one of the following types:

Boolean
String
Integer
Address
Dimension
Position
Gravity_Type

```
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                     Subscript : in Integer;
                     Name      : in String;
                     Value     : in X_Windows.Gravity_Type);
```

Create produces an array of Xt_Arg

```
procedure Create (Arg_List : out Xt_Arg_List;
                 First     : in Integer;
                 Last      : in Integer);
pragma Builtin (Create, 2212);
```

Put sets an element of an array of Xt_Arg

```
procedure Put (Arg_List : out Xt_Arg_List;
              Subscript : in Integer;
              Value      : in Intrinsics.Xt_Ancillary_Types.Xt_Arg);
pragma Builtin (Put, 2213);
```

Get retrieves the value of an element of an array of Xt_Arg

```
function Get
  (Arg_List : in Xt_Arg_List;
   Subscript : in Integer) return Intrinsics.Xt_Ancillary_Types.Xt_Arg;
pragma Builtin (Get, 2214);
```

Get retrieves a slice of an array of Xt_Arg

```
function Get (Arg_List : in Xt_Arg_List;
              First     : in Integer;
              Last      : in Integer) return Xt_Arg_List;
pragma Builtin (Get, 2215);
```

Ada version of the toolkit procedure XtSetValues.

```
procedure Xt_Set_Values (W      : in Intrinsics.Widget;
                       Arg_List : in Xt_Arg_List);
pragma Builtin (Xt_Set_Values, 2216);
```

This set of overloaded functions provides a strongly typed, Ada alternative to the Xt toolkit procedure XtGetValues. Current overloading permits the returned value to be one of the following types:

```

Boolean
Dimension
Font_Record
function Xt_Get_Value (W      : in Intrinsics.Widget;
                      Name : in String) return Boolean;
pragma Builtin (Xt_Get_Value, 2217);

```

This set of overloaded functions provides a strongly typed, Ada alternative to the Xt toolkit procedure XtGetValues. Current overloading permits the returned value to be one of the following types:

```

Boolean
Dimension
Font_Record
function Xt_Get_Value (W      : in Intrinsics.Widget;
                      Name : in String) return Intrinsics.Dimension;
pragma Builtin (Xt_Get_Value, 2218);

```

This set of overloaded functions provides a strongly typed, Ada alternative to the Xt toolkit procedure XtGetValues. Current overloading permits the returned value to be one of the following types:

```

Boolean
Dimension
Font_Record
function Xt_Get_Value (W      : in Intrinsics.Widget;
                      Name : in String) return X_Windows.Fonts.Font_Record;
pragma Builtin (Xt_Get_Value, 2219);

```

This set of overloaded procedures provides a strongly typed, Ada interface to the XtAddCallback toolkit procedure.

Current overloading permits the Closure parameter to be one of the following types:

```

Address
Integer
procedure Xt_Add_Callback (W      : in Intrinsics.Widget;
                          Name : in String;
                          Closure : in Address;
                          Integer : in Integer);

```

```

                                Callback_Proc : in System.Address;
                                Closure       : in System.Address);
pragma Builtin (Xt_Add_Callback, 2220);

```

This set of overloaded procedures provides a strongly typed, Ada interface to the XtAddCallback toolkit procedure.

Current overloading permits the Closure parameter to be one of the following types:

```

    Address
    Integer
    procedure Xt_Add_Callback (W           : in Intrinsics.Widget;
                               Name        : in String;
                               Callback_Proc : in System.Address;
                               Closure     : in Integer);
pragma Builtin (Xt_Add_Callback, 2221);

```

Ada version of the Xt toolkit routine XtCreateWidget.

```

    procedure Xt_Create_Widget (Out_Value : out Intrinsics.Widget;
                               Name       : in String;
                               Widgetclass : in Intrinsics.Widget_Class;
                               Parent      : in Intrinsics.Widget;
                               Arg_List    : in Xt_Arg_List);
pragma Builtin (Xt_Create_Widget, 2222);

```

Ada version of the Xt toolkit routine XtCreateManagedWidget.

```

    procedure Xt_Create_Managed_Widget (Out_Value : out Intrinsics.Widget;
                                         Name     : in String;
                                         Widgetclass : in Intrinsics.Widget_Class;
                                         Parent    : in Intrinsics.Widget;
                                         Arg_List  : in Xt_Arg_List);
pragma Builtin (Xt_Create_Managed_Widget, 2223);

```

Ada version of the Xt toolkit routine XtRealizeWidget.

```

    procedure Xt_Realize_Widget (W : in Intrinsics.Widget);
pragma Builtin (Xt_Realize_Widget, 2224);

```

Ada version of the Xt toolkit routine XtMainLoop.

```

    procedure Xt_Main_Loop;
pragma Builtin (Xt_Main_Loop, 2225);

```

Ada version of the Xt toolkit routine XtDestroyWidget.

```

    procedure Xt_Destroy_Widget (W : in Intrinsics.Widget);
pragma Builtin (Xt_Destroy_Widget, 2226);

```

Ada version of the X library routine XTextWidth.

```
function X_Text_Width (F : in X_Windows.Fonts.Font_Record;  
                      S : in String) return Intrinsics.Dimension;  
pragma Builtin (X_Text_Width, 2227);
```

Ada version of the Xt toolkit routine XtParseTranslationTable.

```
function Xt_Parse_Translation_Table (S : in String) return Xt_Translations;  
pragma Builtin (Xt_Parse_Translation_Table, 2228);
```

Ada version of the Xt toolkit routine XtOverrideTranslations.

```
procedure Xt_Override_Translations (W : in Intrinsics.Widget;  
                                   T : in Xt_Translations);  
pragma Builtin (Xt_Override_Translations, 2229);
```

Ada version of the Xt toolkit routine XtAugmentTranslations.

```
procedure Xt_Augment_Translations (W : in Intrinsics.Widget;  
                                   T : in Xt_Translations);  
pragma Builtin (Xt_Augment_Translations, 2230);
```

Ada version of the Xt toolkit routine XtAddActions.

```
procedure Xt_Add_Actions (Arg_List : in Xt_Arg_List);  
pragma Builtin (Xt_Add_Actions, 2231);
```

Returns a pointer to an interpreted action procedure;

this should be used when providing procedure "addresses" to

Xt_Add_Actions

```
function Action_Procedure_Pointer (Name : in String) return System.Address;  
pragma Builtin (Action_Procedure_Pointer, 2232);
```

Returns a pointer to an interpreted callback procedure;

this should be used when providing procedure "addresses" to

Xt_Add_Callback

```
function Callback_Procedure_Pointer (Name : in String) return System.Address;  
pragma Builtin (Callback_Procedure_Pointer, 2233);
```

Ada version of the Xt toolkit routine _XtDefaultAppContext.

```
function Xt_Default_App_Context return Intrinsics.Xt_App_Context;  
pragma Builtin (Xt_Default_App_Context, 2234);
```

Ada version of the Xt toolkit routine XtAppNextEvent.

```
function Xt_App_Next_Event  
  (App : in Intrinsics.Xt_App_Context) return X_Windows.Events.Event;  
pragma Builtin (Xt_App_Next_Event, 2235);
```

Ada version of the Xt toolkit routine XtDispatchEvent.

```
procedure Xt_Dispatch_Event (E : in X_Windows.Events.Event);
pragma Builtin (Xt_Dispatch_Event 2236);

end Widget_Package;

with System;
with X_Windows;
with Intrinsics;
package body Widget_Package is
  use System;
  use X_Windows;
  use Intrinsics;
  use Xt_Ancillary_Types;

  procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                        Subscript : in Integer;
                        Name       : in String;
                        Value      : in Boolean) is
    Temp_Xt_Arg : Xt_Arg;
  begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
  end Xt_Set_Arg;

  procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                        Subscript : in Integer;
                        Name       : in String;
                        Value      : in String) is
    Temp_Xt_Arg : Xt_Arg;
  begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
  end Xt_Set_Arg;

  procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                        Subscript : in Integer;
                        Name       : in String;
                        Value      : in Integer) is
    Temp_Xt_Arg : Xt_Arg;
  begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
  end Xt_Set_Arg;
```

```

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Address) is
    Temp_Xt_Arg : Xt_Arg;
begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

```

```

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Dimension) is
    Temp_Xt_Arg : Xt_Arg;
begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

```

```

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Position) is
    Temp_Xt_Arg : Xt_Arg;
begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

```

```

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Gravity_Type) is
    Temp_Xt_Arg : Xt_Arg;
begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

```

end Widget_Package;

Xt_Stringdefs is a package which contains commonly used resource name constants.

```
package Xt_Stringdefs is
```

```
-- resource name constants
```

```

Xt_N_Accelerators      : constant String := "accelerators";
Xt_N_Allow_Horiz       : constant String := "allowHoriz";
Xt_N_Allow_Vert        : constant String := "allowVert";
Xt_N_Ancestor_Sensitive : constant String := "ancestorSensitive";
Xt_N_Background        : constant String := "background";
Xt_N_Background_Pixmap : constant String := "backgroundPixmap";
Xt_N_Bitmap            : constant String := "bitmap";
Xt_N_Border_Color      : constant String := "borderColor";
Xt_N_Border            : constant String := "borderColor";
Xt_N_Border_Pixmap     : constant String := "borderPixmap";
Xt_N_Border_Width      : constant String := "borderWidth";
Xt_N_Callback          : constant String := "callback";
Xt_N_Colormap          : constant String := "colormap";
Xt_N_Depth             : constant String := "depth";
Xt_N_Destroy_Callback  : constant String := "destroyCallback";
Xt_N_Edit_Type         : constant String := "editType";
Xt_N_File              : constant String := "file";
Xt_N_Font              : constant String := "font";
Xt_N_Force_Bars        : constant String := "forceBars";
Xt_N_Foreground        : constant String := "foreground";
Xt_N_Function          : constant String := "function";
Xt_N_Height            : constant String := "height";
Xt_N_Highlight         : constant String := "highlight";
Xt_N_H_Space           : constant String := "hSpace";
Xt_N_Index             : constant String := "index";
Xt_N_Inner_Height      : constant String := "innerHeight";
Xt_N_Inner_Width       : constant String := "innerWidth";
Xt_N_Inner_Window      : constant String := "innerWindow";
Xt_N_Insert_Position   : constant String := "insertPosition";
Xt_N_Internal_Height   : constant String := "internalHeight";
Xt_N_Internal_Width    : constant String := "internalWidth";
Xt_N_Jump_Proc         : constant String := "jumpProc";
Xt_N_Justify           : constant String := "justify";
Xt_N_Knob_Height       : constant String := "knobHeight";
Xt_N_Knob_Indent       : constant String := "knobIndent";
Xt_N_Knob_Pixel        : constant String := "knobPixel";
Xt_N_Knob_Width        : constant String := "knobWidth";
Xt_N_Label             : constant String := "label";
Xt_N_Length            : constant String := "length";
Xt_N_Lower_Right       : constant String := "lowerRight";

```

```
Xt_N_Mapped_When_Managed : constant String := "mappedWhenManaged";
Xt_N_Menu_Entry           : constant String := "menuEntry";
Xt_N_Name                 : constant String := "name";
Xt_N_Notify                : constant String := "notify";
Xt_N_Orientation          : constant String := "orientation";
Xt_N_Parameter            : constant String := "parameter";
Xt_N_Pixmap               : constant String := "pixmap";
Xt_N_Popup_Callback       : constant String := "popupCallback";
Xt_N_Popdown_Callback     : constant String := "popdownCallback";
Xt_N_Resize               : constant String := "resize";
Xt_N_Reverse_Video        : constant String := "reverseVideo";
Xt_N_Screen               : constant String := "screen";
Xt_N_Scroll_Proc          : constant String := "scrollProc";
Xt_N_Scroll_D_Cursor      : constant String := "scrollDCursor";
Xt_N_Scroll_H_Cursor      : constant String := "scrollHCursor";
Xt_N_Scroll_L_Cursor      : constant String := "scrollLCursor";
Xt_N_Scroll_R_Cursor      : constant String := "scrollRCursor";
Xt_N_Scroll_U_Cursor      : constant String := "scrollUCursor";
Xt_N_Scroll_V_Cursor      : constant String := "scrollVCursor";
Xt_N_Selection            : constant String := "selection";
Xt_N_Selection_Array       : constant String := "selectionArray";
Xt_N_Sensitive            : constant String := "sensitive";
Xt_N_Shown                : constant String := "shown";
Xt_N_Space                : constant String := "space";
Xt_N_String               : constant String := "string";
Xt_N_Text_Options         : constant String := "textOptions";
Xt_N_Text_Sink            : constant String := "textSink";
Xt_N_Text_Source          : constant String := "textSource";
Xt_N_Thickness           : constant String := "thickness";
Xt_N_Thumb               : constant String := "thumb";
Xt_N_Thumb_Proc           : constant String := "thumbProc";
Xt_N_Top                  : constant String := "top";
Xt_N_Translations         : constant String := "translations";
Xt_N_Update               : constant String := "update";
Xt_N_Use_Bottom           : constant String := "useBottom";
Xt_N_Use_Right            : constant String := "useRight";
Xt_N_Value                : constant String := "value";
Xt_N_V_Space              : constant String := "vSpace";
Xt_N_Width                : constant String := "width";
Xt_N_Window               : constant String := "window";
Xt_N_X                    : constant String := "x";
Xt_N_Y                    : constant String := "y";
```

```
-- resource class constants
```

```

Xt_C_Accelerators      : constant String := "Accelerators";
Xt_C_Background        : constant String := "Background";
Xt_C_Bitmap            : constant String := "Bitmap";
Xt_C_Boolean           : constant String := "Boolean";
Xt_C_Border_Color      : constant String := "BorderColor";
Xt_C_Border_Width      : constant String := "BorderWidth";
Xt_C_Callback          : constant String := "Callback";
Xt_C_Colormap          : constant String := "Colormap";
Xt_C_Color             : constant String := "Color";
Xt_C_Cursor            : constant String := "Cursor";
Xt_C_Depth             : constant String := "Depth";
Xt_C_Edit_Type         : constant String := "EditType";
Xt_C_Event_Bindings    : constant String := "EventBindings";
Xt_C_File              : constant String := "File";
Xt_C_Font              : constant String := "Font";
Xt_C_Foreground        : constant String := "Foreground";
Xt_C_Fraction          : constant String := "Fraction";
Xt_C_Function          : constant String := "Function";
Xt_C_Height            : constant String := "Height";
Xt_C_H_Space           : constant String := "HSpace";
Xt_C_Index             : constant String := "Index";
Xt_C_Insert_Position   : constant String := "InsertPosition";
Xt_C_Interval          : constant String := "Interval";
Xt_C_Justify           : constant String := "Justify";
Xt_C_Knob_Indent       : constant String := "KnobIndent";
Xt_C_Knob_Pixel        : constant String := "KnobPixel";
Xt_C_Label             : constant String := "Label";
Xt_C_Length            : constant String := "Length";
Xt_C_Mapped_When_Managed : constant String := "MappedWhenManaged";
Xt_C_Margin            : constant String := "Margin";
Xt_C_Menu_Entry        : constant String := "MenuEntry";
Xt_C_Notify             : constant String := "Notify";
Xt_C_Orientation       : constant String := "Orientation";
Xt_C_Parameter         : constant String := "Parameter";
Xt_C_Pixmap            : constant String := "Pixmap";
Xt_C_Position          : constant String := "Position";
Xt_C_Resize            : constant String := "Resize";
Xt_C_Reverse_Video     : constant String := "ReverseVideo";
Xt_C_Screen            : constant String := "Screen";
Xt_C_Scroll_Proc       : constant String := "ScrollProc";
Xt_C_Scroll_D_Cursor   : constant String := "ScrollDCursor";
Xt_C_Scroll_H_Cursor   : constant String := "ScrollHCursor";
Xt_C_Scroll_L_Cursor   : constant String := "ScrollLCursor";
Xt_C_Scroll_R_Cursor   : constant String := "ScrollRCursor";
Xt_C_Scroll_U_Cursor   : constant String := "ScrollUCursor";

```

```
Xt_C_Scroll_V_Cursor      : constant String := "ScrollVCursor";
Xt_C_Selection             : constant String := "Selection";
Xt_C_Sensitive             : constant String := "Sensitive";
Xt_C_Selection_Array       : constant String := "SelectionArray";
Xt_C_Space                 : constant String := "Space";
Xt_C_String                : constant String := "String";
Xt_C_Text_Options          : constant String := "TextOptions";
Xt_C_Text_Position         : constant String := "TextPosition";
Xt_C_Text_Sink             : constant String := "TextSink";
Xt_C_Text_Source           : constant String := "TextSource";
Xt_C_Thickness             : constant String := "Thickness";
Xt_C_Thumb                 : constant String := "Thumb";
Xt_C_Translations          : constant String := "Translations";
Xt_C_Value                 : constant String := "Value";
Xt_C_V_Space               : constant String := "VSpace";
Xt_C_Width                 : constant String := "Width";
Xt_C_Window                : constant String := "Window";
Xt_C_X                     : constant String := "X";
Xt_C_Y                     : constant String := "Y";
```

-- resource type constants

```
Xt_R_Accelerator_Table    : constant String := "AcceleratorTable";
Xt_R_Bool                 : constant String := "Bool";
Xt_R_Boolean              : constant String := "Boolean";
Xt_R_Callback             : constant String := "Callback";
Xt_R_Call_Proc            : constant String := "CallProc";
Xt_R_Color                : constant String := "Color";
Xt_R_Cursor               : constant String := "Cursor";
Xt_R_Dimension            : constant String := "Dimension";
Xt_R_Display              : constant String := "Display";
Xt_R_Edit_Mode            : constant String := "EditMode";
Xt_R_File                 : constant String := "File";
Xt_R_Font                 : constant String := "Font";
Xt_R_Font_Struct          : constant String := "FontStruct";
Xt_R_Function             : constant String := "Function";
Xt_R_Geometry             : constant String := "Geometry";
Xt_R_Immediate            : constant String := "Immediate";
Xt_R_Int                  : constant String := "Int";
Xt_R_Justify              : constant String := "Justify";
Xt_R_Long_Boolean         : constant String := "Bool";
Xt_R_Orientation          : constant String := "Orientation";
Xt_R_Pixel                : constant String := "Pixel";
Xt_R_Pixmap               : constant String := "Pixmap";
Xt_R_Pointer              : constant String := "Pointer";
```

```

Xt_R_Position      : constant String := "Position";
Xt_R_Short         : constant String := "Short";
Xt_R_String        : constant String := "String";
Xt_R_String_Table  : constant String := "StringTable";
Xt_R_Unsigned_Char : constant String := "UnsignedChar";
Xt_R_Translation_Table : constant String := "TranslationTable";
Xt_R_Window        : constant String := "Window";

```

```
-- shell specific stringdefs
```

```

Xt_N_Icon_Name     : constant String := "iconName";
Xt_C_Icon_Name     : constant String := "IconName";
Xt_N_Icon_Pixmap   : constant String := "iconPixmap";
Xt_C_Icon_Pixmap   : constant String := "IconPixmap";
Xt_N_Icon_Window   : constant String := "iconWindow";
Xt_C_Icon_Window   : constant String := "IconWindow";
Xt_N_Icon_Mask     : constant String := "iconMask";
Xt_C_Icon_Mask     : constant String := "IconMask";
Xt_N_Window_Group  : constant String := "windowGroup";
Xt_C_Window_Group  : constant String := "WindowGroup";

```

```

Xt_N_Save_Under    : constant String := "saveUnder";
Xt_C_Save_Under    : constant String := "SaveUnder";
Xt_N_Transient     : constant String := "transient";
Xt_C_Transient     : constant String := "Transient";
Xt_N_Override_Redirect : constant String := "overrideRedirect";
Xt_C_Override_Redirect : constant String := "OverrideRedirect";

```

```

Xt_N_Allow_Shell_Resize : constant String := "allowShellResize";
Xt_C_Allow_Shell_Resize : constant String := "AllowShellResize";
Xt_N_Create_Popup_Child_Proc : constant String := "createPopupChildProc";
Xt_C_Create_Popup_Child_Proc : constant String := "CreatePopupChildProc";

```

```

Xt_N_Title : constant String := "title";
Xt_C_Title : constant String := "Title";

```

```
-- The following are only used at creation and can not be changed via
-- SetValues.
```

```

Xt_N_Argc          : constant String := "argc";
Xt_C_Argc          : constant String := "Argc";
Xt_N_Argv          : constant String := "argv";
Xt_C_Argv          : constant String := "Argv";
Xt_N_Icon_X        : constant String := "iconX";
Xt_C_Icon_X        : constant String := "IconX";

```

```

Xt_N_Icon_Y      : constant String := "iconY";
Xt_C_Icon_Y      : constant String := "IconY";
Xt_N_Input       : constant String := "input";
Xt_C_Input       : constant String := "Input";
Xt_N_Iconic      : constant String := "iconic";
Xt_C_Iconic      : constant String := "Iconic";
Xt_N_Initial_State : constant String := "initialState";
Xt_C_Initial_State : constant String := "InitialState";
Xt_N_Geometry     : constant String := "geometry";
Xt_C_Geometry     : constant String := "Geometry";
Xt_N_Min_Width    : constant String := "minWidth";
Xt_C_Min_Width    : constant String := "MinWidth";
Xt_N_Min_Height   : constant String := "minHeight";
Xt_C_Min_Height   : constant String := "MinHeight";
Xt_N_Max_Width    : constant String := "maxWidth";
Xt_C_Max_Width    : constant String := "MaxWidth";
Xt_N_Max_Height   : constant String := "maxHeight";
Xt_C_Max_Height   : constant String := "MaxHeight";
Xt_N_Width_Inc    : constant String := "widthInc";
Xt_C_Width_Inc    : constant String := "WidthInc";
Xt_N_Height_Inc   : constant String := "heightInc";
Xt_C_Height_Inc   : constant String := "HeightInc";
Xt_N_Min_Aspect_Y : constant String := "minAspectY";
Xt_C_Min_Aspect_Y : constant String := "MinAspectY";
Xt_N_Max_Aspect_Y : constant String := "maxAspectY";
Xt_C_Max_Aspect_Y : constant String := "MaxAspectY";
Xt_N_Min_Aspect_X : constant String := "minAspectX";
Xt_C_Min_Aspect_X : constant String := "MinAspectX";
Xt_N_Max_Aspect_X : constant String := "maxAspectX";
Xt_C_Max_Aspect_X : constant String := "MaxAspectX";
Xt_N_Wm_Timeout   : constant String := "wmTimeout";
Xt_C_Wm_Timeout   : constant String := "WmTimeout";
Xt_N_Wait_For_Wm  : constant String := "waitforwm";
Xt_C_Wait_For_Wm  : constant String := "Waitforwm";

end Xt_Stringdefs;

```

7.7.5 Hp_Widgets

```

with Intrinsics;
with Widget_Package;
with System;

```

The package Hp_Widgets is the ACE interface to the Hewlett-Packard

widget set. This package defines all of the numeric constants, enumerated types, and resource names of this widget set, defines all of the widget classes in this set, and supports all of the user widget procedures in this set.

For further information on these routines, see the Hewlett-Packard documentation.

package Hp_Widgets is

```
    use Intrinsic;
    use Widget_Package;
    use Xt_Ancillary_Types;
```

```
    Xw_Single : constant Integer := 0;
    Xw_Multiple : constant Integer := 1;
    Xw_Border : constant Integer := 0;
    Xw_Invert : constant Integer := 1;
    Xw_No_Bias : constant Integer := 0;
    Xw_Row_Bias : constant Integer := 1;
    Xw_Col_Bias : constant Integer := 2;
    Xw_Instant : constant Integer := 0;
    Xw_Sticky : constant Integer := 1;
    Xw_No_Shrink : constant Integer := 0;
    Xw_Shrink_Column : constant Integer := 1;
    Xw_Shrink_All : constant Integer := 2;
    Xw_Auto_Scroll_Off : constant Integer := 0;
    Xw_Auto_Scroll_Horizontal : constant Integer := 1;
    Xw_Auto_Scroll_Vertical : constant Integer := 2;
    Xw_Grow_Off : constant Integer := 0;
    Xw_Grow_Horizontal : constant Integer := 1;
    Xw_Grow_Vertical : constant Integer := 2;
    Xw_N_Of_Many : constant Integer := 0;
    Xw_One_Of_Many : constant Integer := 1;
    Xw_Requested_Columns : constant Integer := 0;
    Xw_Maximum_Columns : constant Integer := 1;
    Xw_Maximum_Unaligned : constant Integer := 2;
    Xw_Right : constant Integer := 0;
    Xw_Left : constant Integer := 1;
    Xw_Center : constant Integer := 2;
    Xw_String : constant Integer := 0;
    Xw_Image : constant Integer := 1;
    Xw_No_Line : constant Integer := 0;
    Xw_Single_Line : constant Integer := 1;
    Xw_Double_Line : constant Integer := 2;
    Xw_Single_Dashed_Line : constant Integer := 3;
    Xw_Double_Dashed_Line : constant Integer := 4;
```

```

Xw_Solid : constant Integer := 0;
Xw_Pattern : constant Integer := 1;
Xw_Transparent : constant Integer := 2;
Xw_Horizontal : constant Integer := 0;
Xw_Vertical : constant Integer := 1;
Xw_Top : constant Integer := 1;
Xw_Bottom : constant Integer := 2;
Xw_Foreground : constant Integer := 0;
Xw_Background : constant Integer := 1;
Xw_25_Foreground : constant Integer := 2;
Xw_50_Foreground : constant Integer := 3;
Xw_75_Foreground : constant Integer := 4;
Xw_Vertical_Tile : constant Integer := 5;
Xw_Horizontal_Tile : constant Integer := 6;
Xw_Slant_Right : constant Integer := 7;
Xw_Slant_Left : constant Integer := 8;
Xw_Highlight_Off : constant Integer := 0;
Xw_Highlight_Enter : constant Integer := 1;
Xw_Highlight_Traversal : constant Integer := 2;
Xw_Arrow_Up : constant Integer := 0;
Xw_Arrow_Down : constant Integer := 1;
Xw_Arrow_Left : constant Integer := 2;
Xw_Arrow_Right : constant Integer := 3;
Xw_Pattern_Border : constant Integer := 1;
Xw_Widget_Defined : constant Integer := 2;
Xw_Ignore : constant Integer := 0;
Xw_Minimize : constant Integer := 1;
Xw_Maximize : constant Integer := 2;

```

```

Tf_No_Fit : constant Integer := 16#01#;
Tf_Include_Tab : constant Integer := 16#02#;
Tf_End_Text : constant Integer := 16#04#;
Tf_Newline : constant Integer := 16#08#;
Tf_Wrap_White_Space : constant Integer := 16#10#;
Tf_Wrap_Any : constant Integer := 16#20#;
Word_Break : constant Integer := 16#01#;
Scroll_Vertical : constant Integer := 16#02#;
Scroll_Horizontal : constant Integer := 16#04#;
Scroll_On_Overflow : constant Integer := 16#08#;
Resize_Width : constant Integer := 16#10#;
Resize_Height : constant Integer := 16#20#;
Editable : constant Integer := 16#40#;

```

```

Xt_N_Traversal_On : constant String := "traversalOn";
Xt_N_Traversal_Type : constant String := "traversalType";

```

Xt_N_Highlight_Style	: constant String := "highlightStyle";
Xt_N_Highlight_Tile	: constant String := "highlightTile";
Xt_N_Highlight_Thickness	: constant String := "highlightThickness";
Xt_N_Highlight_Color	: constant String := "highlightColor";
Xt_N_Background_Tile	: constant String := "backgroundTile";
Xt_N_Cursor	: constant String := "cursor";
Xt_N_Recompute_Size	: constant String := "recomputeSize";
Xt_N_Layout	: constant String := "layout";
Xt_N_Label_Location	: constant String := "labelLocation";
Xt_N_Sensitive_Tile	: constant String := "sensitiveTile";
Xt_N_Columns	: constant String := "columns";
Xt_N_Mode	: constant String := "mode";
Xt_N_Set	: constant String := "set";
Xt_N_Select	: constant String := "select";
Xt_N_Release	: constant String := "release";
Xt_N_Next_Top	: constant String := "nextTop";
Xt_N_Title_Showing	: constant String := "titleShowing";
Xt_N_Mgr_Title_Override	: constant String := "mgrTitleOverride";
Xt_N_Title_Type	: constant String := "titleType";
Xt_N_Title_String	: constant String := "titleString";
Xt_N_Title_Image	: constant String := "titleImage";
Xt_N_Font_Color	: constant String := "fontColor";
Xt_N_Mnemonic	: constant String := "mnemonic";
Xt_N_Underline_Title	: constant String := "underlineTitle";
Xt_N_Mgr_Underline_Override	: constant String := "mgrUnderlineOverride";
Xt_N_Underline_Position	: constant String := "underlinePosition";
Xt_N_Attach_To	: constant String := "attachTo";
Xt_N_Kbd_Accelerator	: constant String := "kbdAccelerator";
Xt_N_Associate_Children	: constant String := "associateChildren";
Xt_N_Menu_Post	: constant String := "menuPost";
Xt_N_Menu_Select	: constant String := "menuSelect";
Xt_N_Post_Accelerator	: constant String := "postAccelerator";
Xt_N_Menu_Unpost	: constant String := "menuUnpost";
Xt_N_Kbd_Select	: constant String := "kbdSelect";
Xt_N_Num_Columns	: constant String := "numColumns";
Xt_N_Row_Position	: constant String := "rowPosition";
Xt_N_Column_Position	: constant String := "columnPosition";
Xt_N_Selection_Method	: constant String := "selectionMethod";
Xt_N_Element_Highlight	: constant String := "elementHighlight";
Xt_N_Selection_Bias	: constant String := "selectionBias";
Xt_N_Selection_Style	: constant String := "selectionStyle";
Xt_N_Column_Width	: constant String := "columnWidth";
Xt_N_Element_Height	: constant String := "elementHeight";
Xt_N_Selected_Elements	: constant String := "selectedElements";
Xt_N_Num_Selected_Elements	: constant String := "numSelectedElements";

Xt_N_Destroy_Mode	: constant String := "destroyMode";
Xt_N_Layout_Type	: constant String := "layoutType";
Xt_N_Force_Size	: constant String := "forceSize";
Xt_N_Single_Row	: constant String := "singleRow";
Xt_N_Separator_Type	: constant String := "separatorType";
Xt_N_Vsb_X	: constant String := "vsbX";
Xt_N_Vsb_Y	: constant String := "vsbY";
Xt_N_Vsb_Width	: constant String := "vsbWidth";
Xt_N_Vsb_Height	: constant String := "vsbHeight";
Xt_N_Hsb_X	: constant String := "hsbX";
Xt_N_Hsb_Y	: constant String := "hsbY";
Xt_N_Hsb_Width	: constant String := "hsbWidth";
Xt_N_Hsb_Height	: constant String := "hsbHeight";
Xt_N_V_Slider_Min	: constant String := "vSliderMin";
Xt_N_V_Slider_Max	: constant String := "vSliderMax";
Xt_N_V_Slider_Origin	: constant String := "vSliderOrigin";
Xt_N_V_Slider_Extent	: constant String := "vSliderExtent";
Xt_N_H_Slider_Min	: constant String := "hSliderMin";
Xt_N_H_Slider_Max	: constant String := "hSliderMax";
Xt_N_H_Slider_Origin	: constant String := "hSliderOrigin";
Xt_N_H_Slider_Extent	: constant String := "hSliderExtent";
Xt_N_H_Scroll_Event	: constant String := "hScrollEvent";
Xt_N_V_Scroll_Event	: constant String := "vScrollEvent";
Xt_N_V_Scroll_Bar_Width	: constant String := "vScrollBarWidth";
Xt_N_V_Scroll_Bar_Height	: constant String := "vScrollBarHeight";
Xt_N_H_Scroll_Bar_Width	: constant String := "hScrollBarWidth";
Xt_N_H_Scroll_Bar_Height	: constant String := "hScrollBarHeight";
Xt_N_Force_Vertical_Sb	: constant String := "forceVerticalSB";
Xt_N_Force_Horizontal_Sb	: constant String := "forceHorizontalSB";
Xt_N_Initial_X	: constant String := "initialX";
Xt_N_Initial_Y	: constant String := "initialY";
Xt_N_Border_Pad	: constant String := "borderPad";
Xt_N_S_Rimage	: constant String := "rasterImage";
Xt_N_Show_Selected	: constant String := "showSelected";
Xt_N_Display_Position	: constant String := "displayPosition";
Xt_N_Insert_Position	: constant String := "insertPosition";
Xt_N_Left_Margin	: constant String := "leftMargin";
Xt_N_Right_Margin	: constant String := "rightMargin";
Xt_N_Top_Margin	: constant String := "topMargin";
Xt_N_Bottom_Margin	: constant String := "bottomMargin";
Xt_N_Selection_Array	: constant String := "selectionArray";
Xt_N_Text_Source	: constant String := "textSource";
Xt_N_Selection	: constant String := "selection";
Xt_N_Maximum_Size	: constant String := "maximumSize";
Xt_N_Edit_Type	: constant String := "editType";

```

Xt_N_File                : constant String := "file";
-- Xt_N_String : constant String := "string"; -- defined in Xt_Stringdefs
Xt_N_Length : constant String := "length";
-- Xt_N_Font : constant String := "font"; -- defined in Xt_Stringdefs
Xt_N_Disk_Src            : constant String := "disksrc";
Xt_N_String_Src          : constant String := "stringsrc";
Xt_N_Execute             : constant String := "execute";
Xt_N_Source_Type         : constant String := "sourceType";
Xt_N_Motion_Verification : constant String := "motionVerification";
Xt_N_Modify_Verification : constant String := "modifyVerification";
Xt_N_Leave_Verification   : constant String := "leaveVerification";
Xt_N_Wrap : constant String := "wrap";
Xt_N_Wrap_Form           : constant String := "wrapForm";
Xt_N_Wrap_Break          : constant String := "wrapBreak";
Xt_N_Scroll              : constant String := "scroll";
Xt_N_Grow : constant String := "grow";
Xt_N_Alignment           : constant String := "alignment";
Xt_N_Line_Space          : constant String := "lineSpace";
Xt_N_Gravity             : constant String := "gravity";
Xt_N_Slider_Min          : constant String := "sliderMin";
Xt_N_Slider_Max          : constant String := "sliderMax";
Xt_N_Slider_Origin       : constant String := "sliderOrigin";
Xt_N_Slider_Extent       : constant String := "sliderExtent";
Xt_N_Slider_Color        : constant String := "sliderColor";
Xt_N_Slide_Orientation   : constant String := "slideOrientation";
Xt_N_Slide_Move_Type     : constant String := "slideMoveType";
Xt_N_Slide_Area_Tile     : constant String := "slideAreaTile";
Xt_N_Slider_Moved        : constant String := "sliderMoved";
Xt_N_Area_Selected       : constant String := "areaSelected";
Xt_N_Slider_Tile         : constant String := "sliderTile";
Xt_N_Slider_Released     : constant String := "sliderReleased";
Xt_N_X_Ref_Name          : constant String := "xRefName";
Xt_N_X_Ref_Widget        : constant String := "xRefWidget";
Xt_N_X_Offset            : constant String := "xOffset";
Xt_N_X_Add_Width         : constant String := "xAddWidth";
Xt_N_X_Vary_Offset       : constant String := "xVaryOffset";
Xt_N_X_Resizable         : constant String := "xResizable";
Xt_N_X_Attach_Right      : constant String := "xAttachRight";
Xt_N_X_Attach_Offset     : constant String := "xAttachOffset";
Xt_N_Y_Ref_Name          : constant String := "yRefName";
Xt_N_Y_Ref_Widget        : constant String := "yRefWidget";
Xt_N_Y_Offset            : constant String := "yOffset";
Xt_N_Y_Add_Height       : constant String := "yAddHeight";
Xt_N_Y_Vary_Offset       : constant String := "yVaryOffset";
Xt_N_Y_Resizable         : constant String := "yResizable";

```

Xt_N_Y_Attach_Bottom	: constant String := "yAttachBottom";
Xt_N_Y_Attach_Offset	: constant String := "yAttachOffset";
Xt_N_Pixel_Scale	: constant String := "pixelScale";
Xt_N_Grid_Thickness	: constant String := "gridThickness";
Xt_N_Image	: constant String := "image";
Xt_N_Draw_Color	: constant String := "drawColor";
Xt_N_Erase_Color	: constant String := "eraseColor";
Xt_N_Erase_On	: constant String := "eraseOn";
Xt_N_Label_Type	: constant String := "labelType";
Xt_N_Label_Image	: constant String := "labelImage";
Xt_N_Cascade_Image	: constant String := "cascadeImage";
Xt_N_Mark_Image	: constant String := "markImage";
Xt_N_Set_Mark	: constant String := "setMark";
Xt_N_Cascade_On	: constant String := "cascadeOn";
Xt_N_Invert_On_Enter	: constant String := "invertOnEnter";
Xt_N_Mgr_Override_Mnemonic	: constant String := "mgrOverrideMnemonic";
Xt_N_Cascade_Select	: constant String := "cascadeSelect";
Xt_N_Cascade_Unselect	: constant String := "cascadeUnselect";
Xt_N_Menu_Mgr_Id	: constant String := "menuMgrId";
Xt_N_Scrollbar_Orientation	: constant String := "scrollbarOrientation";
Xt_N_Selection_Color	: constant String := "selectionColor";
Xt_N_Initial_Delay	: constant String := "initialDelay";
Xt_N_Repeat_Rate	: constant String := "repeatRate";
Xt_N_Granularity	: constant String := "granularity";
Xt_N_Invert_On_Select	: constant String := "invertOnSelect";
Xt_N_Toggle	: constant String := "toggle";
Xt_N_Square	: constant String := "square";
Xt_N_Select_Color	: constant String := "selectColor";
Xt_N_Allow_Resize	: constant String := "allowResize";
Xt_N_Sash_Indent	: constant String := "sashIndent";
Xt_N_Refigure_Mode	: constant String := "refigureMode";
Xt_N_Padding	: constant String := "padding";
Xt_N_Min	: constant String := "min";
Xt_N_Max	: constant String := "max";
Xt_N_Skip_Adjust	: constant String := "skipAdjust";
Xt_N_Framed	: constant String := "framed";
Xt_N_Border_Frame	: constant String := "borderFrame";
Xt_N_Expose	: constant String := "expose";
Xt_N_Resize	: constant String := "resize";
Xt_N_Key_Down	: constant String := "keyDown";
Xt_N_Sticky_Menus	: constant String := "stickyMenus";
Xt_N_Allow_Cascades	: constant String := "allowCascades";
Xt_N_Pulldown_Bar_Id	: constant String := "pulldownBarId";
Xt_N_Strip	: constant String := "strip";
Xt_N_Title_Precedence	: constant String := "titlePrecedence";

```

Xt_N_Title_Foreground      : constant String := "titleForeground";
Xt_N_Title_Background      : constant String := "titleBackground";
Xt_N_Title_Region          : constant String := "titleRegion";
Xt_N_Title_Position        : constant String := "titlePosition";
Xt_N_Title_Rpadding        : constant String := "titleRPadding";
Xt_N_Title_Lpadding        : constant String := "titleLPadding";
Xt_N_Title_Border_Width    : constant String := "titleBorderWidth";
Xt_N_Title_Translations    : constant String := "titleTranslations";
Xt_N_Title_Hspace          : constant String := "titleHSpace";
Xt_N_Title_Vspace          : constant String := "titleVSpace";
Xt_N_Title_Select          : constant String := "titleSelect";
Xt_N_Title_Release         : constant String := "titleRelease";
Xt_N_Title_Next_Top        : constant String := "titleNextTop";
Xt_N_Titlebar_Tile         : constant String := "titlebarTile";
Xt_N_Enter                 : constant String := "enter";
Xt_N_Leave                  : constant String := "leave";
Xt_N_Region                : constant String := "region";
Xt_N_Position              : constant String := "position";
Xt_N_L_Padding             : constant String := "lPadding";
Xt_N_R_Padding             : constant String := "rPadding";
Xt_N_Precedence            : constant String := "precedence";
Xt_N_Title_To_Menu_Pad     : constant String := "titleToMenuPad";
Xt_N_Work_Space_To_Sibling_Pad : constant String := "workSpaceToSiblingPad";
Xt_N_Widget_Type           : constant String := "widgetType";
Xt_N_Top_Level             : constant String := "topLevel";
Xt_N_Display_Title         : constant String := "displayTitle";
Xt_N_Causes_Resize         : constant String := "causesResize";
Xt_N_Arrow_Direction       : constant String := "arrowDirection";

```

```

subtype Xw_Text_Position is Cardinal;

```

```

type Xw_Text_Source_Ptr is new System.Address;

```

```

type Xw_Text_Sink_Ptr is new System.Address;

```

```

type Xw_Alignment is
  (Xw_Align_None,
   Xw_Align_Left,
   Xw_Align_Center,
   Xw_Align_Right);

```

```

type Xw_Widget_Type is (Xw_Unknown, Xw_Pulldown, Xw_Title, Xw_Work_Space);

```

```

type Xw_Scan_Direction is (Xw_Sd_Left, Xw_Sd_Right);

```

```

type Xw_Scan_Type is

```

```
(Xw_St_Positions,  
  Xw_St_White_Space,  
  Xw_St_Eol,  
  Xw_St_Last);  
  
type Xw_Edit_Type is (Xw_Text_Read, Xw_Text_Append, Xw_Text_Edit);  
  
type Xw_Edit_Result is  
  (Xw_Edit_Done,  
   Xw_Edit_Error,  
   Xw_Edit_Pos_Error,  
   Xw_Edit_Reject);  
  
type Xw_Verify_Op_Type is (Motion_Verify, Mod_Verify, Leave_Verify);  
  
type Xw_Source_Type is (Xw_String_Src, Xw_Disk_Src, Xw_Prog_Defined_Src);  
  
type Xw_Wrap is (Xw_Wrap_Off, Xw_Soft_Wrap, Xw_Hard_Wrap);  
  
type Xw_Wrap_Form is (Xw_Source_Form, Xw_Display_Form);  
  
type Xw_Wrap_Break is (Xw_Wrap_Any, Xw_Wrap_White_Space);  
  
function Xw_Arrow_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Arrow_Widget_Class, 2301);  
function Xw_Bulletin_Board_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Bulletin_Board_Widget_Class, 2302);  
function Xw_Bulletin_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Bulletin_Widget_Class, 2303);  
function Xw_Button_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Button_Widget_Class, 2304);  
function Xw_Cascade_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Cascade_Widget_Class, 2305);  
function Xw_Form_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Form_Widget_Class, 2306);  
function Xw_Image_Edit_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Image_Edit_Widget_Class, 2307);  
function Xw_List_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_List_Widget_Class, 2308);  
function Xw_Listrow_Col_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Listrow_Col_Widget_Class, 2309);  
function Xw_Manager_Widget_Class return Intrinsics.Widget_Class;  
pragma Builtin (Xw_Manager_Widget_Class, 2310);  
function Xw_Menu_Button_Widget_Class return Intrinsics.Widget_Class;
```

```
pragma Builtin (Xw_Menu_Button_Widget_Class, 2311);
function Xw_Menu_Sep_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Menu_Sep_Widget_Class, 2312);
function Xw_Menubutton_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Menubutton_Widget_Class, 2313);
function Xw_Menumgr_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Menumgr_Widget_Class, 2314);
function Xw_Menupane_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Menupane_Widget_Class, 2315);
function Xw_Panel_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Panel_Widget_Class, 2316);
function Xw_Popup_Mgr_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Popup_Mgr_Widget_Class, 2317);
function Xw_Popupmgr_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Popupmgr_Widget_Class, 2318);
function Xw_Primitive_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Primitive_Widget_Class, 2319);
function Xw_Push_Button_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Push_Button_Widget_Class, 2320);
function Xw_Row_Col_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Row_Col_Widget_Class, 2321);
function Xw_Sash_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Sash_Widget_Class, 2322);
function Xw_Scroll_Bar_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Scroll_Bar_Widget_Class, 2323);
function Xw_Scrollbar_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Scrollbar_Widget_Class, 2324);
function Xw_Scrolled_Window_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Scrolled_Window_Widget_Class, 2325);

function Xw_Sraster_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Sraster_Widget_Class, 2326);
function Xw_Static_Raster_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Static_Raster_Widget_Class, 2327);
function Xw_Static_Text_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Static_Text_Widget_Class, 2328);
function Xw_Statictext_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Statictext_Widget_Class, 2329);
function Xw_Swindow_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Swindow_Widget_Class, 2330);
function Xw_Text_Edit_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Text_Edit_Widget_Class, 2331);
function Xw_Textedit_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Textedit_Widget_Class, 2332);
```

```

function Xw_Title_Bar_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Title_Bar_Widget_Class, 2333);
function Xw_Titlebar_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Titlebar_Widget_Class, 2334);
function Xw_Toggle_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Toggle_Widget_Class, 2335);

-- Using Xw_V_Paned_Widget_Class creates a demand for a procedure
-- called _XtCreateFontCursor, which does not seem to be in any
-- X library (rhp, 5/2/90)
-- Aha! This is apparently a misprint for _XCreateFontCursor (rhp)

-- function Xw_V_Paned_Widget_Class return Intrinsics.Widget_Class;
-- pragma Builtin (Xw_V_Paned_Widget_Class, 2336);

function Xw_Valuator_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Valuator_Widget_Class, 2337);
function Xw_Work_Space_Widget_Class return Intrinsics.Widget_Class;
pragma Builtin (Xw_Work_Space_Widget_Class, 2338);

Ada interface to the widget procedure XwTextClearBuffer
  procedure Xw_Text_Clear_Buffer (W : in Widget);
  pragma Builtin (Xw_Text_Clear_Buffer, 2401);

Ada interface to the widget procedure XwTextCopyBuffer
  function Xw_Text_Copy_Buffer (W : in Widget) return String;
  pragma Builtin (Xw_Text_Copy_Buffer, 2402);

Ada interface to the widget procedure XwTextCopySelection
  function Xw_Text_Copy_Selection (W : in Widget) return String;
  pragma Builtin (Xw_Text_Copy_Selection, 2403);

Ada interface to the widget procedure XwTextReadSubString
  procedure Xw_Text_Read_Sub_String (W           : in Widget;
                                     Start_Pos    : in Integer;
                                     End_Pos      : in Integer;
                                     Target       : out String;
                                     Target_Used  : out Integer;
                                     Source_Used  : out Integer);
  pragma Builtin (Xw_Text_Read_Sub_String, 2404);

Ada interface to the widget procedure XwTextUnsetSelection
  procedure Xw_Text_Unset_Selection (W : in Widget);
  pragma Builtin (Xw_Text_Unset_Selection, 2405);

```

Ada interface to the widget procedure XwTextSetSelection

```
procedure Xw_Text_Set_Selection (W      : in Widget;
                                Left   : in Xw_Text_Position;
                                Right  : in Xw_Text_Position);
pragma Builtin (Xw_Text_Set_Selection, 2406);
```

Ada interface to the widget procedure XwTextReplace

```
function Xw_Text_Replace (W      : in Widget;
                          Start_Pos : in Xw_Text_Position;
                          End_Pos   : in Xw_Text_Position;
                          S        : in String) return Xw_Edit_Result;
pragma Builtin (Xw_Text_Replace, 2407);
```

Ada interface to the widget procedure XwTextRedraw

```
procedure Xw_Text_Redraw (W : in Widget);
pragma Builtin (Xw_Text_Redraw, 2408);
```

Ada interface to the widget procedure XwTextUpdate

```
procedure Xw_Text_Update (W      : in Widget;
                          Status : in Boolean);
pragma Builtin (Xw_Text_Update, 2409);
```

Ada interface to the widget procedure XwTextInsert

```
procedure Xw_Text_Insert (W : in Widget;
                          S : in String);
pragma Builtin (Xw_Text_Insert, 2410);
```

Ada interface to the widget procedure XwTextGetLastPos

```
function Xw_Text_Get_Last_Pos (W : in Widget) return Xw_Text_Position;
pragma Builtin (Xw_Text_Get_Last_Pos, 2411);
```

Ada interface to the widget procedure XwTextGetSelectionPos

```
procedure Xw_Text_Get_Selection_Pos (W      : in Widget;
                                     Left   : out Xw_Text_Position;
                                     Right  : out Xw_Text_Position);
pragma Builtin (Xw_Text_Get_Selection_Pos, 2412);
```

Ada interface to the widget procedure XwTextSetInsertPos

```
procedure Xw_Text_Set_Insert_Pos (W      : in Widget;
                                  Pos     : in Xw_Text_Position);
pragma Builtin (Xw_Text_Set_Insert_Pos, 2413);
```

Ada interface to the widget procedure XwTextGetInsertPos

```
function Xw_Text_Get_Insert_Pos (W : in Widget) return Xw_Text_Position;  
pragma Builtin (Xw_Text_Get_Insert_Pos, 2414);
```

Ada interface to the widget procedure XwTextSetSource

```
procedure Xw_Text_Set_Source (W           : in Widget;  
                             Source      : in Xw_Text_Source_Ptr;  
                             Start_Pos   : in Xw_Text_Position);  
pragma Builtin (Xw_Text_Set_Source, 2415);
```

Ada interface to the widget procedure XwAsciiSinkCreate

```
function Xw_Ascii_Sink_Create  
  (W      : in Widget;  
   Args : in Xt_Arg_List) return Xw_Text_Sink_Ptr;  
pragma Builtin (Xw_Ascii_Sink_Create, 2416);
```

Ada interface to the widget procedure XwDiskSourceCreate

```
function Xw_Disk_Source_Create  
  (W      : in Widget;  
   Args : in Xt_Arg_List) return Xw_Text_Source_Ptr;  
pragma Builtin (Xw_Disk_Source_Create, 2417);
```

Ada interface to the widget procedure XwDiskSourceDestroy

```
procedure Xw_Disk_Source_Destroy (Src : in Xw_Text_Source_Ptr);  
pragma Builtin (Xw_Disk_Source_Destroy, 2418);
```

Ada interface to the widget procedure XwStringSourceCreate

```
function Xw_String_Source_Create  
  (W      : in Widget;  
   Args : in Xt_Arg_List) return Xw_Text_Source_Ptr;  
pragma Builtin (Xw_String_Source_Create, 2419);
```

Ada interface to the widget procedure XwStringSourceDestroy

```
procedure Xw_String_Source_Destroy (Src : in Xw_Text_Source_Ptr);  
pragma Builtin (Xw_String_Source_Destroy, 2420);
```

Ada interface to the widget procedure XwMoveFocus

```
procedure Xw_Move_Focus (W : in Widget);  
pragma Builtin (Xw_Move_Focus, 2421);
```

This set of overloaded procedures provides a strongly typed,
Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one
of the enumerated types defined in this package, namely:

Xw_Alignment

```

Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg   : out Xt_Arg;
                      Name  : in String;
                      Value  : in Xw_Alignment);
pragma Builtin (Xt_Set_Arg, 2422);

```

This set of overloaded procedures provides a strongly typed,

Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List   : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name    : in String;
                      Value    : in Xw_Alignment);

```

This set of overloaded procedures provides a strongly typed,

Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type

```

```

Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg   : out Xt_Arg;
                      Name   : in String;
                      Value  : in Xw_Widget_Type);
pragma Builtin (Xt_Set_Arg, 2423);

```

This set of overloaded procedures provides a strongly typed,
Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one
of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                     Subscript : in Integer;
                     Name      : in String;
                     Value     : in Xw_Widget_Type);

```

This set of overloaded procedures provides a strongly typed,
Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one
of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap

```

```

Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg   : out Xt_Arg;
                      Name   : in String;
                      Value  : in Xw_Scan_Direction);
pragma Builtin (Xt_Set_Arg, 2424);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value     : in Xw_Scan_Direction);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg   : out Xt_Arg;
                      Name   : in String;

```

```

                                Value : in Xw_Scan_Type);
pragma Builtin (Xt_Set_Arg, 2425);

```

This set of overloaded procedures provides a strongly typed,
Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one
of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value     : in Xw_Scan_Type);

```

This set of overloaded procedures provides a strongly typed,
Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one
of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg      : out Xt_Arg;
                      Name     : in String;
                      Value    : in Xw_Edit_Type);
pragma Builtin (Xt_Set_Arg, 2426);

```

This set of overloaded procedures provides a strongly typed,

Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```
Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
```

```
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Xw_Edit_Type);
```

This set of overloaded procedures provides a strongly typed,

Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```
Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
```

```
procedure Xt_Set_Arg (Arg      : out Xt_Arg;
                      Name     : in String;
                      Value    : in Xw_Edit_Result);
pragma Builtin (Xt_Set_Arg, 2427);
```

This set of overloaded procedures provides a strongly typed,

Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```
Xw_Alignment
```

```

Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Xw_Edit_Result);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg      : out Xt_Arg;
                      Name     : in String;
                      Value    : in Xw_Verify_Op_Type);
pragma Builtin (Xt_Set_Arg, 2428);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type

```

```

Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value     : in Xw_Verify_Op_Type);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg      : out Xt_Arg;
                      Name     : in String;
                      Value    : in Xw_Source_Type);
pragma Builtin (Xt_Set_Arg, 2429);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap

```

```

Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value      : in Xw_Source_Type);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg. The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg      : out Xt_Arg;
                      Name     : in String;
                      Value     : in Xw_Wrap);
pragma Builtin (Xt_Set_Arg, 2430);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg. The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;

```

```

Name      : in String;
Value     : in Xw_Wrap);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg   : out Xt_Arg;
                      Name   : in String;
                      Value  : in Xw_Wrap_Form);
pragma Builtin (Xt_Set_Arg, 2431);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg.

The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value     : in Xw_Wrap_Form);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg. The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (Arg    : out Xt_Arg;
                      Name   : in String;
                      Value  : in Xw_Wrap_Break);
pragma Builtin (Xt_Set_Arg, 2432);

```

This set of overloaded procedures provides a strongly typed, Ada version of the C macro XtSetArg. The overloading provided here permits the Value parameter to be one of the enumerated types defined in this package, namely:

```

Xw_Alignment
Xw_Widget_Type
Xw_Scan_Direction
Xw_Scan_Type
Xw_Edit_Type
Xw_Edit_Result
Xw_Verify_Op_Type
Xw_Source_Type
Xw_Wrap
Xw_Wrap_Form
Xw_Wrap_Break
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value     : in Xw_Wrap_Break);

```

end Hp_Widgets;

```

with Intrinsic;
with Widget_Package;

```

```
package body Hp_Widgets is
  use Widget_Package;
  use Intrinsic;
  use Xt_Ancillary_Types;

  procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                        Subscript : in Integer;
                        Name       : in String;
                        Value      : in Xw_Alignment) is
    Temp_Xt_Arg : Xt_Arg;
  begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
  end Xt_Set_Arg;

  procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                        Subscript : in Integer;
                        Name       : in String;
                        Value      : in Xw_Widget_Type) is
    Temp_Xt_Arg : Xt_Arg;
  begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
  end Xt_Set_Arg;

  procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                        Subscript : in Integer;
                        Name       : in String;
                        Value      : in Xw_Scan_Direction) is
    Temp_Xt_Arg : Xt_Arg;
  begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
  end Xt_Set_Arg;

  procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                        Subscript : in Integer;
                        Name       : in String;
                        Value      : in Xw_Scan_Type) is
    Temp_Xt_Arg : Xt_Arg;
  begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
  end Xt_Set_Arg;
```

```
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Xw_Edit_Type) is
    Temp_Xt_Arg : Xt_Arg;
begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Xw_Edit_Result) is
    Temp_Xt_Arg : Xt_Arg;
begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Xw_Verify_Op_Type) is
    Temp_Xt_Arg : Xt_Arg;
begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Xw_Source_Type) is
    Temp_Xt_Arg : Xt_Arg;
begin
    Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
    Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name       : in String;
                      Value      : in Xw_Wrap) is
    Temp_Xt_Arg : Xt_Arg;
```

```

begin
  Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
  Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value     : in Xw_Wrap_Form) is
  Temp_Xt_Arg : Xt_Arg;
begin
  Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
  Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

procedure Xt_Set_Arg (List      : in out Xt_Arg_List;
                      Subscript : in Integer;
                      Name      : in String;
                      Value     : in Xw_Wrap_Break) is
  Temp_Xt_Arg : Xt_Arg;
begin
  Xt_Set_Arg (Temp_Xt_Arg, Name, Value);
  Put (List, Subscript, Temp_Xt_Arg);
end Xt_Set_Arg;

end Hp_Widgets;

```

8 More Information About Some Ace Features

This section provides a detailed look at some Ace ADTs, to help the user become familiar with the functionality they provide.

8.1 Xt Toolkit Interface

The Xt toolkit is a library of functions that provide an object-oriented approach to the X windowing system. The toolkit consists of a set of basic procedures, called *intrinsic*s and a set of facilities for creating window objects called *widgets*. Widgets are individual representatives of widget classes. While the Xt toolkit permits the creation of new widget classes, many sets of widget classes are already available in the public domain.

ACE provides an Ada interface to a subset of the Xt intrinsic functions and to a complete set of widget classes, namely, those offered in the Hewlett-Packard widget set. This section

is not intended as a detailed description of either the Xt intrinsics or the Hewlett-Packard widget set, for which appropriate documentation is available from the respective distributors. This section describes only those features of the ACE interface that differ from a compiled interface to these libraries.

8.1.1 Xt Prototyping Sessions

The Xt toolkit assumes that an application will be driven entirely from the X windows that it creates. Each Xt application consists, therefore, of two distinct parts. The initialization code creates the application's windows, panels, and pushbuttons. After the initialization code has run, control is given to an endless event-handling loop. The event loop responds to events such as pointer motions and keyboard input by calling user-defined response code through a mechanism known as *callback*. In the Xt toolkit's view of an application, this process continues as long as the application. That is, when the windowing part of the application is finished, so is the application program itself.

In order to provide this view of an application without requiring ACE to be restarted, the ACE interface provides a means of creating separate Xt sessions. Each Xt session appears to the Xt toolkit as a new application program.

8.1.1.1 Starting an Xt Prototyping Session. To begin a new Xt session, the ACE user should use the routine *Xt_Initialize*. When the Xt session begins, ACE will signal the user by changing its prompt from

ACE>

to

ACE Xt>

Calling *Xt_Initialize* not only begins an Xt session, but also performs the toolkit functions associated with this procedure. For this reason, it is an error to call this procedure once the Xt session has started, since this would appear to the Xt toolkit as if an application were trying to initialize twice, a violation of the Xt toolkit's rules.

Also, if the *Xt_Initialize* procedure is called from some other procedure, the code following it will be interpreted twice, once in the Xt session, and again after the Xt session is finished. If the following code contains certain calls to the Xt toolkit, the toolkit's error action, taken when these calls are performed outside the Xt prototyping session, can cause ACE to exit. It is therefore a good practice not to call this procedure as part of some other procedure (unless calling *Xt_Initialize* is the only instruction, as shown in the example below).

The Ada specification of the *Xt_Initialize* procedure is given below.

```

procedure Xt_Initialize
  (Out_Value : out Intrinsics.Widget;
   Name      : in String;
   Classname : in String;
   Urlist    : in X_Windows.Resource_Manager.Xrm_Option_Desc_List;
   Num_Urs   : in Intrinsics.Cardinal;
   Argc      : in out Intrinsics.Cardinal;
   Argv      : in out Renamed_Xlib_Types.Argv_Type);

```

After the *Xt_Initialize* procedure has been called, the *Out_Value* parameter will contain the value of the shell widget, which is used as the ultimate ancestor for all other widgets created during the Xt prototyping session.

Because the *Xt_Initialize* procedure requires a complex set of arguments, it will be convenient to create an argument-free procedure that calls it. For the reasons given above, the call to *Xt_Initialize* should be the only instruction in such a procedure. A typical version of such a procedure is shown below.

```

procedure Start_Xt is
  Argc : Intrinsics.Cardinal := 0;
  Argv : Renamed_Xlib_Types.Argv_Type := 0;
begin
  Xt_Initialize (Shell_Widget, "", "",
                Widget_Package.Null_Xrm_Option_List, 0,
                Argc, Argv);
end Start_Xt;

```

Currently, ACE does not provide any means of creating objects of either the type *Xrm_Option_Desc_L* or the type *Argv_Type*. Therefore, the values shown in this example for the parameters *Urlist*, *Num_Urs*, *Argc*, and *Argv* are the only values allowable.

8.1.1.2 Finishing an Xt Prototyping Session To end an Xt prototyping session, the user should cause ACE to interpret the instruction

```

exit ACE;

```

That is, the same instruction that is normally used to end an ACE run will also end an Xt prototyping session. When used from such a session, however, ACE will not end, but will return to its normal mode of operation, with its original user prompt.

This instruction will normally be part of a callback procedure. A typical means of ending an application that uses the Hewlett-Packard widget set is to incorporate a pushbutton widget whose selection callback is a procedure that performs this instruction.

In the current version of ACE, any callback routine that fails to execute properly, that is, any routine that raises an exception, will also cause the Xt prototyping session to end.

8.1.2 Xt Argument Lists

Most of the Xt routines are parameterized by lists of pairs, with each pair consisting of a resource name and a resource value. Because ACE does not support arrays, special routines are provided to manipulate such lists. The Ada specifications of these routines are shown below.

```

procedure Create (Arg_List : out Xt_Arg_List;
                  First     : in Integer;
                  Last      : in Integer);

procedure Put (Arg_List : out Xt_Arg_List;
              Subscript  : in Integer;
              Value      : in Xt_Arg);

function Get
  (Arg_List : in Xt_Arg_List;
   Subscript : in Integer) return Xt_Arg;

function Get (Arg_List : in Xt_Arg_List;
              First     : in Integer;
              Last      : in Integer) return Xt_Arg_List;
```

The *Create* procedure reserves space for an argument list and assigns upper and lower bounds. The *Put* procedure assigns a value to a specified member of an argument list. The *Get* function with a single *Integer* parameter returns the value of a member of an argument list, while the *Get* function with two *Integer* parameters returns a specified slice.

To create values of the type *Xt_Arg* suitable for inclusion in argument lists, ACE provides the Xt toolkit procedure *Xt_Set_Arg*. Since the elements of argument lists are of many different types, the *Xt_Set_Arg* procedure is overloaded many times. All of its overloaded versions, however, conform to one of two basic patterns.

In the first pattern, *Xt_Set_Arg* is used to create a single value of type *Xt_Arg*, as shown below.

```

procedure Xt_Set_Arg (Arg    : out Xt_Arg;
                     Name    : in String;
                     Value   : in Some_Value_Type);
```

In the second pattern, an argument value is built and inserted directly into an argument list, as shown below.

```
procedure Xt_Set_Arg (List      : in out Xt_Arg_List;  
                      Subscript : in Integer;  
                      Name      : in String;  
                      Value     : in Some_Value_Type);
```

The reader should consult Xt toolkit documentation for the allowable resource names and value types.

8.1.3 Xt Callbacks

The Xt toolkit supports two kinds of user response procedures, called *callback* procedures and *action* procedures. From the programmer's point of view, there is little difference between these two kinds of procedures. Both kinds are registered with the toolkit during the initialization part of an Xt application and both are called by the event-handling loop in response to some set of events.

8.1.3.1 Callback Procedures. Callback procedures are associated with specific callback points provided by each widget class. Callback procedures are registered through the procedure *Xt_Add_Callback*, whose Ada specification is given below.

```
procedure Xt_Add_Callback (W           : in Intrinsics.Widget;  
                          Name        : in String;  
                          Callback_Proc : in System.Address;  
                          Closure      : in Integer);
```

To obtain a value for the *Callback_Proc* parameter, the ACE user should use the function *Callback_Procedure_Pointer*, which accepts a callback procedure's name, in the form of a string, and returns a value that ACE can use as the callback procedure's address. The *Closure* parameter is an arbitrary value that will be passed to the callback procedure when it is called. The typical use of this parameter is to permit a callback routine to be registered more than once and to distinguish the two registrations.

Every callback procedure must conform (except for the names of the procedure and its parameters) to the following specification.

```
procedure Typical_Callback (W           : in Intrinsics.Widget;  
                           Client_Data : in Integer;  
                           Call_Data   : in Integer);
```

When the procedure is called, the second parameter (*Client_Data*) will be a copy of the *Closure* parameter given at registration. The meaning of the third parameter (*Call_Data*) is defined by each widget class.

Callback procedures should be either independent compilation units or should be at the top level of some package. That is, callback procedures should not be nested inside other procedures. The results of registering and invoking nested callback procedures are unpredictable.

8.1.3.2 Action Procedures. Action procedures can be associated with various events, such as pointer motions and keyboard input, that are not specific to a single widget class. The use of action procedures involves two distinct steps. First, the procedures are registered with the Xt toolkit by using the *Xt_Add_Actions* procedure. The *Xt_Add_Actions* procedure takes a single argument of type *Xt_Arg_List*. The elements of this list consist of pairs whose first element is a string containing an arbitrary symbolic name for the action procedure and whose second element is a procedure pointer.

To obtain values for the required procedure pointers, the ACE user should use the function *Action_Procedure_Pointer*, which accepts an action procedure's name, in the form of a string, and returns a value that ACE can use as the action procedure's address.

Every action procedure must conform (except for the names of the procedure and its parameters) to the following specification.

```
procedure Typical_Action (W      : in Intrinsics.Widget;
                          E      : in X_Windows.Events.Event;
                          Args   : in X_Windows.String_List);
```

Since the current version of ACE does not provide support for manipulating the *Event* and *String_List* types, the second and third parameters are not currently useful, but they must nevertheless be declared.

After symbolic names have been assigned to action procedures by *Xt_Add_Actions*, these symbolic names are used by an Xt toolkit facility called *translation*, whose task is to translate the symbolic names of the action procedures and the (predefined) symbolic names of events into the proper widget structures. The relevant Xt routines that cause these things to happen are *Xt_Parse_Translation_Table*, *Xt_Override_Translations*, and *Xt_Augment_Translations*. For further information on these routines, the user should consult Xt toolkit documentation, since ACE does not impose any of its own constraints on their use.

Action procedures should be either independent compilation units or should be at the top level of some package. That is, action procedures should not be nested inside other procedures. The results of registering and invoking nested action procedures are unpredictable.

For implementation reasons, no more than thirty-two action procedures may be registered by any one Xt prototyping session. Calling the function *Action_Procedure_Pointer* more than

thirty-two times in one Xt prototyping session will produce an error message and will not register the action procedure. There is no similar limit on the number of callback procedures, however.

8.1.3.3 Callback Interpretation. Because callback and action procedures are actually interpreted by ACE, there is more flexibility in ACE callbacks than in compiled code. In particular, the parameter supplied to the *Callback_Procedure_Pointer* and *Action_Procedure_Pointer* functions is not confined to simply the names of valid callback and action procedures, but may be any valid sequence of Ada statements *ending* with such a name.

Since this has no equivalent in compiled code, the ACE user should use this feature very sparingly; otherwise, the eventual transition to a compiled application will be more difficult.

This feature does have one important application in ACE, however. It permits input events to be associated with ACE's debugging pragmas. For example, if the ACE user wants a pushbutton that will cause ACE's trace facility to start, he can use

```
Xt_Add_Callback
  (Button_Widget, Xt_N_Select,
   Callback_Procedure_Pointer ("pragma TRACE (On); No_Op"), 0);
```

where *No_Op* is a callback procedure that does nothing.

8.1.3.4 Prototyping Callbacks. The usual method for prototyping Xt code in ACE will be to test each version of the code in a separate prototyping session. Because ACE callbacks are interpreted, however, callback code can be changed without ending the prototyping session.

This does require special care on the part of the ACE user. In a usual Xt application, the initialization part of the application ends by calling the procedure *Xt_Main_Loop*, which is available in the ACE interface. This procedure is an endless event-handling loop, which does not end until the application itself is finished.

The following procedure, which performs exactly the same functions as *Xt_Main_Loop*, can also be entered in the ACE Xt interface.

```
procedure My_Main_Loop is
  E : Event;
  C : Intrinsics.Xt_App_Context := Xt_Default_App_Context;
begin
  loop
    E := Xt_App_Next_Event (C);
```

```
Xt_Dispatch_Event (E);  
end loop;  
end My_Main_Loop;
```

The advantage of using such a loop instead of *Xt_Main_Loop*, however, is that the routine shown above can be interrupted and resumed. While this loop is running, the ACE interrupt key will cause it to stop, and the ACE *Continue* procedure will cause it to resume.

While the loop is interrupted, the user is free to change the contents of a callback or action procedure. Thus, such procedures can be altered without ending the prototyping session.

The user should be prepared for some peculiar effects when using this facility. While the event loop is interrupted, nothing will be handling the events for the X_t application. This means, for example, that if the application window is covered and re-exposed, it will not be refreshed until the loop resumes.

The user should also be aware that the *Xt_App_Next_Event* function does not return until an event occurs. In practice, this means that the ACE interrupt key will not take effect until after some event occurs in the X_t application. The typical sequence of operations for interrupting this loop, therefore, will be:

- Move the mouse pointer to the ACE window
- Enter the interrupt key
- Move the mouse pointer to an X_t application widget (to cause an event)
- Return the pointer to the ACE window (to enter new callback code)

8.1.4 Transition to Compiled Code

The purpose of the ACE interface to the X_t toolkit is to permit the prototyping of applications that will eventually be compiled. For this reason, great care has been taken to make the ACE interface nearly the same as the compilable Ada bindings to the X_t toolkit. Once a given application has been successfully prototyped in ACE, it should be possible to compile the same code with very few changes. A few changes will be necessary, however, which are noted below.

- *Procedure Pointers*

References to the functions *Callback_Procedure_Pointer* and *Action_Procedure_Pointer* should be replaced by an appropriate compilable means of obtaining a procedure pointer. For some X_t bindings, this will be the Ada *ADDRESS* attribute, while others will require procedure pointers to be obtained by generic instantiation.

As noted above, there is no compilable equivalent to ACE's ability to create procedure pointers from a sequence of Ada statements. Parts of the application that use this ability must be either eliminated or rewritten.

- *Array References*

The use of ACE procedures that eliminate array references should be replaced by the appropriate subscripted expressions or slices.

In particular, the overloadings of the *Xt_Set_Arg* procedure that take a list and a subscript as separate arguments will not be available in compilable Xt bindings; these should be replaced by a single subscripted argument.

Also, the *Get* function that returns a slice of an *Xt_Arg_List* will not be available, and should be replaced by array slice expressions. Such slices will usually be arguments to the procedures *Xt_Add_Actions*, *Xt_Create_Widget*, and *Xt_Create_Managed_Widget*.

- *Ending the Application*

Since the statement *exit ACE;* has no equivalent in compiled Ada, it must be replaced by some other means of ending the application. This can take the form of interfacing to the host computer's process termination routine, or can be implemented by raising an unhandled exception.

8.1.5 A Small Example

The following is an example of how the ACE Xt interface might be used. Once this example has been entered, it is run in the following way:

- Run the procedure *Start_Xt*.
- When the prompt changes to the Xt prototyping prompt, run the procedure *Aloha*.
- The screen should now show a small panel with two buttons labelled "Hello" and "Goodbye". Pressing the left mouse button with the pointer in the "Hello" button will cause the application to print "Hello"; pressing the "Goodbye" button will cause the application to print "Goodbye" and end the prototyping session.

Not all of the Ada *use* statements shown below are actually needed to run this example. They will be required, however, to extend it.

```
use System;  
use Strings;  
use Text_IO;  
use X_Windows;  
use Fonts;
```

```
use Events;
use Resource_Manager;
use Renamed_Xlib_Types;
use Intrinsics;
use Xt_Ancillary_Types;
use Widget_Package;
use Xt_Stringdefs;
use Hp_Widgets;

Shell_Widget : Widget;

procedure Start_Xt is
  Argc : Cardinal := 0;
  Argv : Argv_Type := 0;
begin
  Xt_Initialize (Shell_Widget, "", "", Null_Xrm_Option_List, 0, Argc, Argv);
end Start_Xt;

procedure Message (W           : in Widget;
                   Client_Data : in Integer;
                   Call_Data   : in Integer) is
begin
  if Client_Data = 0 then
    Put_Line ("Hello");
  else
    Put_Line ("Goodbye");
    exit Ace;
  end if;
end Message;

procedure Aloha is

  My_Xt_Arg_List : Xt_Arg_List;
  Panel          : Widget;
  Button1        : Widget;
  Button2        : Widget;

begin
  Create (My_Xt_Arg_List, 1, 20);
  Xt_Create_Managed_Widget (Panel, "", Xw_Row_Col_Widget_Class,
                           Shell_Widget, Null_Xt_Arg_List);
  Xt_Set_Arg (My_Xt_Arg_List, 1, Xt_N_Label, "Hello");
  Xt_Create_Managed_Widget
    (Button1, "", Xw_Push_Button_Widget_Class, Panel,
     Get (My_Xt_Arg_List, 1, 1));
```

```

Xt_Add_Callback (Button1, Xt_N_Select,
                  Callback_Procedure_Pointer ("Message"), 0);
Xt_Set_Arg (My_Xt_Arg_List, 1, Xt_N_Label, "Goodbye");
Xt_Create_Managed_Widget
  (Button2, "", Xw_Push_Button_Widget_Class, Panel,
   Get (My_Xt_Arg_List, 1, 1));
Xt_Add_Callback (Button2, Xt_N_Select,
                  Callback_Procedure_Pointer ("Message"), 1);
Xt_Realize_Widget (Shell_Widget);
Xt_Main_Loop;
end Aloha;

```

8.2 The *Key_Bindings* Package

The package, *Key_Bindings*, provides the user with Ada routines to edit command line input, and recall earlier lines. It also supplies the mechanism to invoke a specified Ada procedure by a user designated keystroke. These capabilities can greatly enhance the day-to-day environment of a development programmer.

Ace is able to provide these capabilities because it now maintains a log of the input (within the bounds of the user specified limit), which is available to the history routines declared in *Key_Bindings*, and finds the binding associated with each keystroke (or sequence of keystrokes) and calls the routine indicated by the binding. Usually that will be *Do_Self_Insert*. But there are many other things one might want to do, and the code to accomplish some of those things is already written, and available to the user at the touch of a key, for the paltry price of a call to *Make_Binding*.

8.2.1 Making Bindings

The package *Key_Bindings* contains the procedures that will associate a routine with a keystroke, or sequence thereof, and also all the empowered binding choices (command names). When Ace is started a set of default bindings is installed. Each printable ASCII character (from space to tilde) is bound to *Do_Self_Insert*. Carriage return and line feed are bound to *Line_Separator*. The user's environment is checked to find out which key the user normally uses for deleting characters and that key is bound to *Delete*.

It is necessary to always have at least one character bound to *Line_Separator*, and the *Make_Binding* procedures enforce this rule by refusing to cooperate with a request to change the binding of the last thing currently bound to *Line_Separator*. This requirement exists because the *Line_Separator* binding is the only signal to Ace that a line of input has been completed.

In order to obtain the services of any of the other provided routines the user must make the appropriate call to *Make_Binding*. The specifications for the two versions of *Make_Binding*

are:

```
procedure Make_Binding (C           : in Character;  
                        Binding      : in Commands;  
                        Optional_String : in String := "");  
  
procedure Make_Binding (Char_Seq    : in String;  
                        Binding      : in Commands;  
                        Optional_String : in String := "");
```

The first parameter of these procedures should be the value transmitted when the relevant key (or sequence) is struck.

The legal choices for the *Binding* parameter are:

Line_Separator
Interpret_String
History_Back
History_Forward
Move_Left
Move_Right
Beginning_Of_Line
End_Of_Line
Start_Of_History
End_Of_History
Delete
Delete_This_Char
Show_History
Show_History_Limit
Set_History_Limit
Kill_Line
Quoted_Insert
Refresh_Current_Line
Refresh_Current_Line_And_Prompt
Self_Insert

The third parameter is needed only when the second parameter is set to *Interpret_String*, which is explained below. Please note that it is meaningless to bind a function or procedure with parameters to a keystroke, because there is no way to communicate the return value to other parts of the code, nor to pass parameter values to the procedure. However, such routines can be manipulated through the use of *Interpret_String*.

8.2.1.1 Using *Interpret_String*. In addition to the provided routines the user can write Ada procedures of their own and bind them to a key via the *Interpret_String* procedure. To use this facility, the user should simply provide the procedure to be invoked and a binding for *Interpret_String*. For example:

```
Make_Binding (Ascii.Esc & "[226z",
              Key_Bindings.Interpret_String,
              "My_Procedure;");
```

The third parameter is a string containing an Ada statement, exactly as it would appear in a unit of Ada code. Usually this statement will be a call to some routine, but it can be more. Some possible values are:

- "Flag := Some_Function;"
- "Some_Procedure (Arg1, Arg2);"
- "Var := 0; Other_Procedure; Other_Var := Something;"

Any variables appearing in such a string must be global and must be declared before the bound key is struck. This hypothetical routine, *My_Procedure*, in general, is just like any other routine written in Ace. However, its I/O must be done according to the guidelines given below.

8.2.1.2 Intermixing I/O with *Key_Bindings* Routines. If the user wishes to include screen or keyboard oriented I/O, they may use *Text_IO*. However, the programmer must keep in mind that their procedure will be executed within the internal environment of Ace which has put the terminal in raw mode. Therefore it is necessary to place calls around the *Text_IO* routines, to reset the terminal before *Text_IO* is used, and then to set it back again before returning to Ace's control. The same is true when using *Key_Bindings* routines, which assume that they know what the current line looks like and that the terminal is in raw mode.

In addition to setting the terminal back to raw mode, the user must always call either *Refresh_Current_Line_And_Prompt* or *Rewrite_Current_Line_And_Prompt*, when entering the *Ace_Command_Line_Input* or *Key_Bindings* environment. This routine will restore the screen so that the active line on the screen looks as it did before the user hit the key that caused *My_Procedure* to be invoked, (or restore a different value as the current line of input if the programmer prefers). The programmer should be certain that the cursor is positioned at the beginning of a line before making the call to restore the line. This is because, currently, there is no way for the *Key_Bindings* routines to clear away output they didn't write, so you must be at the beginning of a line in order to ensure that the restored line looks ok. In a later release this requirement should go away.

The prohibition against making calls to *Text_IO* routines, while in the cultural context of Ace or the *Key_Bindings* package, also applies in reverse: *Key_Bindings* routines (other than *Terminal_Set*) should not be called in the *Text_IO* (cooked) environment. This is because

those routines assume they know how the active line on the screen looks. To summarize, calls to *Text_Io* routines can be intermingled with *Key_Bindings* calls just so long as:

- you are in cooked mode when making a *Text_Io* call
- you are in raw mode when returning to Ace or when making a *Key_Bindings* call
- if the terminal is reset to cooked mode, then a call to restore the active line on the screen must be made, after *Terminal_Set*, and before any *Key_Bindings* call or return to Ace

When control enters *My_Procedure*, the terminal will be in raw (set) mode. And after a call to any of the four Refresh/Rewrite procedures, the terminal will be in raw mode. To explicitly flip from one mode to the other, call *Terminal_Set* to gain the *Ace_Command_Line_Input* or *Key_Bindings* environment, or *Terminal_Reset* to prepare for calls to *Text_Io* routines.

Of course if *My_Procedure* doesn't have any screen output or keyboard input then the terminal will stay set and *Key_Bindings* routines can be freely interspersed with other Ada code. All the *Key_Bindings* routines listed in the Bindings section can be used in *My_Procedure*, with the single exception of *Interpret_String*.

8.2.2 An Example for *Interpret_String*

Here is an example of what *My_Procedure* might look like:

```
with Text_Io;
with Ace_Integer_Io;
with Strings;
with Key_Bindings; use Key_Bindings;

-- This procedure copies the input typed so far,
-- a user supplied number of times

procedure My_Procedure is
  Line      : String (1 .. 1024);
  Length    : Natural;
  The_Integer : Integer := - 1;
  I         : Integer := 1;
begin
  Do_Get_Current_Line (Line, Length);
  Do_Kill_Line;
  Terminal_Reset; -- go to cooked mode
  Text_Io.Put ("How many times should this pattern be repeated?");
```

```
Ace_Integer_Io.Get (The_Integer);  
Terminal_Set; -- go to raw  
-- the carriage return the user ended the integer with  
-- prepares us for the insert  
while I <= The_Integer loop  
    Do_Insert_String (Strings.Slice( Line, 1, Length));  
    I := I + 1;  
end loop;  
end My_Procedure;
```

9 Supported Ada Features

The command language of ACE is Ada. This section describes the language features of Ada that are currently supported by the ACE prototype. (Section 5 described the pragmas supported by ACE.) The Ada features implemented in the prototype version of ACE are those that are necessary to make use of Ada as a command language, those that are needed to provide a basis for the construction of abstract data types within the environment, and those that together form a logical collection of Ada functionality. This feature set includes packages, subprograms, subprogram derivation and overload resolution, assignment, conditional, iteration, termination, and return statements, and scalar types with type and subtype declarations.

The following list summarizes the Ada language features currently implemented in ACE. The numbering scheme corresponds to that of the Ada Reference Manual. Qualifications on the level of support, if any, are given following the appropriate section. Support is not provided for those sections of the reference manual that are omitted from this list.

As development is continuing for ACE, an explicit "Not Yet Supported" message is produced for those Ada features that are still being developed. However, a range of possibilities exist for the outcome when an incomplete feature of Ada within ACE is invoked. These possibilities include the generation of an Internal Error message, no error message or result due to the fact that the statement is parsed successfully and/or discarded, and successful execution of some portion of a partially implemented Ada feature. The following list should be consulted to verify that only supported features of Ada have been exercised.

*Ada Language Reference Manual Sections
supported by ACE*

2. Lexical Elements

2.1 Character Set

2.2 Lexical Elements, Separators, & Delimiters

2.3 Identifiers

2.4 Numeric Literals

2.4.1 Decimal Literals

2.4.2 Based Literals

2.5 Character Literals

Partial support for Character Literals is provided

2.6 String Literals

Partial support for String Literals is provided

2.7 Comments

2.8 Pragmas

2.9 Reserved Words

3. Declarations & Types

3.1 Declarations

3.2 Objects & Named Numbers

3.2.1 Object Declarations

3.3 Types & Subtypes

3.3.1 Type Declarations

No support for *incomplete_type_definition* and *private_type_definition* declarations

3.3.2 Subtype Declarations

No support for *constraints*

3.4 Derived Types

No support for *constraint* portion of *subtype_indication*

3.5.1 Enumeration Types

No support for *character_literal* as enumeration literal

No support for overloaded enumeration literals

3.5.3 Boolean Types

3.5.4 Integer Types

Support only for predefined *Integer*

3.6.3 The Type String

Not supported as a one-dimensional array;

Temporarily implemented as a developer's string type—*Ace_String*

3.9 Declarative Parts

4. Names & Expressions

4.1 Names

4.2 Literals

4.4 Expressions

4.5 Operators & Expression Evaluation

4.5.1 Logical Operators & Short-circuit Control Forms

Short-circuit control forms (*and then*, *or else*) not yet supported

4.5.2 Relational Operators & Membership Tests

Membership tests (*in*) not yet supported

4.5.3 Binary Adding Operators

4.5.4 Unary Adding Operators

4.5.5 Multiplying Operators

4.5.6 Highest Precedence Operators

4.6 Type Conversions

4.9 Static Expressions & Static Subtypes

4.10 Universal Expressions

Partial support for *universal_integer*

5. Statements

5.1 Simple & Compound Statements - Sequence of Statements

No support for statement *label*

5.2 Assignment Statement

5.3 If Statement

5.4 Case Statement

others, list of choices not yet supported

5.5 Loop Statement

for iteration scheme not yet supported

loop simple name not yet supported

5.7 Exit Statement

loop name not yet supported

5.8 Return Statement

6. Subprograms

6.1 Subprogram Declarations

6.2 Formal Parameter Modes

6.3 Subprogram Bodies

Exceptions not yet supported, except for exception declarations

6.3.1 Conformance Rules

Legal variations in subprogram conformance not yet supported (numeric literals, simple vs. expanded name, string literal as operator symbol)

6.4 Subprogram Calls

Named association not yet supported

6.4.1 Parameter Associations

6.5 Function Subprograms

6.6 Parameter & Result Type Profile - Overloading of Subprograms

Complete overload resolution not yet supported (e.g. overloaded subprogram where an actual parameter is also overloaded, overloaded functions that differ only in return type)

6.7 Overloading of Operators

Check that proper number of parameters are given for the overloaded operator symbol used as a function designator not yet supported

7. Packages

7.1 Package Structure

7.2 Package Specifications & Declarations

7.3 Package Bodies

exception part of a package not yet supported

8. Visibility Rules

Partial support of expanded names is provided

8.1 Declarative Region

Declarative blocks are partially supported

8.2 Scope of Declarations

8.3 Visibility

8.4 Use Clauses

8.6 The Package Standard

8.7 The Context of Overload Resolution

Partial support for overload resolution is provided

10. Program Structure & Compilation Issues

10.1 Compilation Units - Library Units

10.3 Order of Compilation

10.5 Elaboration of Library Units

10.6 Program Optimization

14. Input-Output

14.1 External Files and File Objects

14.3 Text Input-Output

14.3.1 File Management

14.3.2 Default Input and Output Files

14.3.3 Specification of Line and Page Lengths

14.3.4 Operations on Columns, Lines, and Pages

14.3.5 Get and Put Procedures

14.3.6 Input-Output of Characters and Strings

14.3.7 Input-Output for Integer Types

Support for the predefined type *Integer* provided by the ACE package *Ace_Integer_IO*

14.3.10 Specification of the Package `Text_IO`

14.4 Exceptions in Input-Output

Exceptions not supported, except for exception declarations

14.5 Specification of the Package `IO_Exceptions`

Exceptions not supported, except for exception declarations

10 Examples

This section provides some condensed examples of interactive sessions with ACE. These examples include illustrations of Ada being used as an interactive command language, manipulating Ada subprograms and packages as data, and interfacing with the underlying host operating system. The use of the abstract data types described in previous sections, along with the Ada statements supported by this ACE prototype, are illustrated in these examples.

10.1 Interactive Ada Example

% ACE

Ada Command Environment; Revision: 6.0 Date: 89/12/14 14:51:30

Processing environment definition...

ACE>

ACE> -- EXAMPLE INTERACTIVE ACE SESSION

ACE> -- This is a simple example of the development of Ada subprograms

ACE> -- that shows the use of ACE for interactive development.

ACE> -- This example makes use of the packages Text_IO and

ACE> -- ACE_Integer_IO of the ACE environment. It also shows examples

ACE> -- of declarations, procedures, functions, overload resolution,

ACE> -- the if statement, loop statement, return statement, assignment

ACE> -- statement, use statement and exit statement, and comments.

ACE>

ACE> use text_io;

ACE> use ace_integer_io;

ACE>

ACE> -- Develop a Put_Line routine for Integers

ACE> procedure Put_Line (Item : in Integer) is

| begin

| Put (Item);

| New_Line;

| end Put_Line;

ACE>

ACE> i : integer;

ACE> i := 100;

ACE> Put_Line (i);

100

ACE>

ACE>

ACE> -- Develop a simple Put routine for Booleans

ACE> procedure Put (Item : in Boolean) is

| begin

| Put (Boolean'Image(item));

```
| end Put;
ACE>
ACE> b : boolean := false;
ACE> Put (b); New_Line;
FALSE
ACE>
ACE> -- Develop a routine to determine whether a number is prime
ACE> function Is_Prime (n : in integer) return Boolean is
|   local : integer := 3;
|   begin
|     if n rem 2 = 0 then
|       return False;
|     end if;
|
|     while local ** 2 <= n loop
|       if n rem local = 0 then
|         return False;
|       end if;
|       local := local + 2;
|     end loop;
|
|     return True;
|   end Is_Prime;
ACE>
ACE> put (Is_Prime (7)); New_Line;
TRUE
ACE> put (Is_Prime (143)); New_Line;
FALSE
ACE>
ACE>
ACE> exit ace;
Exiting ACE
```

10.2 Manipulating Ada Components

% ACE

Ada Command Environment; Revision: 6.0 Date: 89/12/14 14:51:30

Processing environment definition...

```
ACE>
ACE> -- Example of iterative development of an Ada subprogram
ACE> -- followed by incorporation of the subprogram into a
ACE> -- package that is stored to an external file.
ACE>
```

```

ACE> -- Develop a Factorial routine
ACE> function Fact (n : in integer) return integer is
| begin
|   if n = 0 then
|     return 1;
|   else
|     -- >>> ERROR: will type + rather than * <<<
|     return n + fact (n-1);
|   end if;
| end;
ACE>
ACE>
ACE> put (fact (4));
Error: could not find a valid subprogram specification: put
ACE>
ACE> -- The above error is due to the fact that we have not yet
ACE> -- acquired direct visibility of the Put operation from the
ACE> -- package Ace_Integer_Io within Text_Io.
ACE>
ACE> -- Use these packages
ACE> use text_io;
ACE> use ace_integer_io;
ACE>
ACE> put (fact (4)); New_Line;
    11
ACE>
ACE> -- Fact produces an erroneous result.
ACE> -- Invoke the edit and re-interpret ADT to fix the
ACE> -- erroneous statement in the Fact function.
ACE>
ACE> edit_and_interpret ("fact");
Spawning a window for editing /sun/ace/src/fact.tmp

    -- At this point, the editor is brought up and the
    -- "+" is changed to a "*". The changes are written
    -- out and the editor is terminated.

ACE> -- Display the revised Fact function
ACE> list ("fact", both);
function fact (n : in integer) return integer;
-- Form => " 819"

function fact (n : in integer) return integer is
-- Form => " 819"
begin

```

```
    if (n = 0) then
        return 1;
    else
        return (n * fact((n - 1)));
    end if;
end fact;
```

```
ACE> Put (Fact (4)); New_Line;
      24
```

```
ACE>
```

```
ACE> Put (Fact (5)); New_Line;
      120
```

```
ACE>
```

```
ACE> -- Fact routine is now working.
```

```
ACE> -- Define a package into which Fact will be placed.
```

```
ACE>
```

```
ACE> package Math_Routines is
| end;
```

```
ACE>
```

```
ACE>
```

```
ACE> -- Using the Program_Unit abstract data types and operations,
```

```
ACE> -- define a package object and a subprogram object.
```

```
ACE>
```

```
ACE> package_object : package_type;
```

```
ACE> function_object : subprogram_type;
```

```
ACE>
```

```
ACE> -- Associate the package object with Math_Routines;
```

```
ACE> -- associate the function object with Fact
```

```
ACE>
```

```
ACE> open (package_object, out_prog_unit, "Math_Routines");
```

```
ACE> open (function_object, in_prog_unit, "Fact");
```

```
ACE>
```

```
ACE> -- Place the function object into the package object,
```

```
ACE> -- with the implementation of the function being placed
```

```
ACE> -- in the body of the package and visible in the package
```

```
ACE> -- specification.
```

```
ACE>
```

```
ACE> put (function_object, package_object, in_body, visible);
```

```
ACE>
```

```
ACE> -- Display the package object, both spec and body
```

```
ACE>
```

```
ACE> list (package_object, both);
```

```
package math_routines is
```

```
-- Form => " 825"
```

```
function fact (n : in integer) return integer;

end math_routines;
package body math_routines is

function fact (n : in integer) return integer is

begin
    if (n = 0) then
        return 1;
    else
        return (n * fact((n - 1)));
    end if;
end fact;

end math_routines;
ACE>
ACE> -- Store the Math_Routines package in an external file
ACE>
ACE> put_file ( Name(package_object), "", "math_package.ada");
ACE>
ACE> -- End of ACE session
ACE> exit ace;
```

10.3 Interfacing with Host O/S (UNIX)

% ACE

Ada Command Environment; Revision: 6.0 Date: 89/12/14 14:51:30

Processing environment definition...

```
ACE> -- Example of interacting with external Ada compilation
ACE> -- system and the underlying host operating system
ACE>
ACE> -- Define a program text object
ACE> program_object : program_text;
ACE>
ACE> -- List out the contents of the current working directory
ACE> list;
```

```
Test_Cases
a.out
ada.lib
find_primes.a
ACE>
```

```
requirements
math_package.ada
test_data
```

```
ACE> -- Associate the program text object with the file
ACE> -- find_primes.a
ACE>
ACE> open (program_object, in_object, "find_primes.a");
ACE>
ACE> -- List out the contents of the program_object
ACE>
ACE> list (program_object);
with Text_Io;
procedure Find_Primes is

    function Is_Prime (n : in integer) return Boolean is
        local : integer := 3;
    begin
        if n rem 2 = 0 then
            return False;
        end if;

        while local ** 2 <= n loop
            if n rem local = 0 then
                return False;
            end if;
            local := local + 2;
        end loop;

        return True;
    end Is_Prime;

    package Int_Io is new Text_Io.Integer_Io (Integer);

begin
    Text_Io.Put_Line ("PRIME NUMBERS FROM 2 .. 64");
    for i in 2 .. 64 loop
        if Is_Prime (i) then
            Int_Io.Put (i);
            Text_Io.Put_Line (" is a prime number");
        end if;
    end loop;
end Find_Primes;
ACE>
ACE> -- Submit the program text object to the external Ada
ACE> -- batch compilation system (Verdix VADS), where
ACE> -- Find_Primes is the name of the main compilation unit
ACE> -- and the executable image will be stored into the file
ACE> -- "find_primes.exe"
```

ACE>

ACE> compile (program_object, "Find_Primes", "find_primes.exe");
Compiling -M Find_Primes -o find_primes.exe /johndoe/find_primes.a

ACE>

ACE> -- Define a program binary object and execute it

ACE>

ACE> load_image : program_binary;

ACE> open (load_image, in_object, "find_primes.exe");

ACE>

ACE> execute (load_image);

PRIME NUMBERS FROM 2 .. 64

3 is a prime number

5 is a prime number

7 is a prime number

11 is a prime number

13 is a prime number

17 is a prime number

19 is a prime number

23 is a prime number

29 is a prime number

31 is a prime number

37 is a prime number

41 is a prime number

43 is a prime number

47 is a prime number

53 is a prime number

59 is a prime number

61 is a prime number

ACE>

ACE> -- End of ACE session

ACE> exit ace;

Exiting ACE

Index

- "*" pkg. def. 25
- "***" pkg. def. 25
- "+" pkg. def. 25, 30
- "-" pkg. def. 25, 30
- "/" pkg. def. 25
- "/=" pkg. def. 24-25, 28
- "i" pkg. def. 24-25, 28, 30
- "i=" pkg. def. 24-25, 28, 30
- "=" pkg. def. 24-25, 28
- "i" pkg. def. 24-25, 28, 30
- "i=" pkg. def. 24-25, 28, 30
- "abs" pkg. def. 25
- "and" pkg. def. 24
- "mod" pkg. def. 25
- "not" pkg. def. 24
- "or" pkg. def. 24
- "rem" pkg. def. 25
- "xor" pkg. def. 24
- "&" pkg. def. 28
- abstract data types (ADT) 2-3, 8, 12-17, 23, 109
- abstract data types and a command language 13
- Ace Adt 37
- Ace Integer Io 33
- Ace User X Window System 19, 57
- Ace X Window System 19, 56
- ACE 4
- ace_ada 4
- Ace_Adt 19
- ace_edit 4
- Ace_Integer_Io 18
- Ace_Io 18
- Ace_Standard 18
- Action_Procedure_Pointer pkg. def. 75
- Action_Procedure_Pointer 114
- Ada language standard 1, 13
- Ada-oriented environment 18
- ADT Philosophy 12
- Append_To_Cais_Arg_List pkg. def. 60
- applications 12
- Ascent pkg. def. 64
- ASCII 18
- Binary Files 16
- Binary Objects 19, 47
- bindings.ace 23
- Break pkg. def. 39
- built-in 17
- Cais Routines 19, 23, 59
- CAIS-A 59
- cais.ace 23
- Cais_Node_Kinds 59
- Cais_Relationship_Kinds 59
- Calendar 18, 30
- Callback_Procedure_Pointer pkg. def. 75
- Character Io 33
- Check_Style pkg. def. 61
- Clear_Break pkg. def. 39
- Clear_Window pkg. def. 56
- Clock pkg. def. 30, 63
- Close pkg. def. 32, 42, 47
- Close 14
- Col pkg. def. 33
- Column control 32
- command applicability 15
- command environment 13
- command extensibility 16
- command language 1, 15-16, 23, 123
- command language commands 20, 36
- command line editing 23
- command line history 23
- command specialization 15
- command structure 14
- commands.ace 23
- compilation units 2
- Compile pkg. def. 37, 45
- complete command 4
- consistency 14
- consistent objects and operations 15
- Continue pkg. def. 39
- Continue_Prompt 4, 7
- Copy pkg. def. 41
- Copy 16
- Count_Features pkg. def. 61
- Count_Statements pkg. def. 61
- Cpu Time 20, 23, 63

cpu_time.ace 23
 Create pkg. def. 32, 42, 47, 57, 72, 112
 create ACE 5
 Create 14-16
 Create_Cais_Argument_List pkg. def. 60
 Create_Window pkg. def. 56
 Current_Ace_Input pkg. def. 35
 Current_Ace_Input 3
 Current_Ace_Output pkg. def. 35
 Current_Directory pkg. def. 43
 Current_Input pkg. def. 32
 Current_Output pkg. def. 32
 data abstraction 12-13
 Day pkg. def. 30
 Debugger 7-8, 19, 39
 declarations 2
 Default Io files 32
 Delete pkg. def. 32, 37-38, 47-48, 57
 Delete 14, 16
 Delete_File pkg. def. 41
 derived subprograms 15
 Descent pkg. def. 64
 Destroy_Window pkg. def. 56
 Deuse pkg. def. 38, 48
 development approach 13
 Diana_Browser pkg. def. 61
 Diana_Cleanlib pkg. def. 61
 Diana_Create_Predefined_Env pkg. def. 62
 Diana_Front_End pkg. def. 61
 Diana_Make_Predefined_Env pkg. def. 62
 Diana_Mklib pkg. def. 61
 Diana_Rmlib pkg. def. 61
 Difference pkg. def. 63
 Directory Files 16
 Directory Objects 19, 42
 Display pkg. def. 39
 Display_Current_Statement pkg. def. 39
 Display_Next pkg. def. 39
 Display_Previous pkg. def. 39
 Do_Beg_Of_Line pkg. def. 54
 Do_Delete pkg. def. 54
 Do_Delete.This_Char pkg. def. 54
 Do_Get_Current_Character pkg. def. 55
 Do_Get_Current_Column pkg. def. 55
 Do_Get_Current_Line pkg. def. 54
 Do_Goto_End_Of_History pkg. def. 54
 Do_Goto_End_Of_Line pkg. def. 54
 Do_Goto_Start_Of_History pkg. def. 54
 Do_History_Back pkg. def. 54
 Do_History_Forward pkg. def. 54
 Do_Insert_Ascii pkg. def. 54
 Do_Insert_String pkg. def. 54
 Do_Interpret_String pkg. def. 54
 Do_Kill_Line pkg. def. 54
 Do_Move_Left pkg. def. 54
 Do_Move_Right pkg. def. 54
 Do_Quoted_Insert pkg. def. 54
 Do_Refresh_Current_Line pkg. def. 54
 Do_Refresh_Current_Line_And_Prompt pkg. def. 54
 Do_Rewrite_Current_Line pkg. def. 54
 Do_Rewrite_Current_Line_And_Prompt pkg. def. 54
 Do_Self_Insert pkg. def. 54
 Do_Set_History_Limit pkg. def. 54
 Do_Show_History pkg. def. 54
 Do_Show_History_Limit pkg. def. 54
 Draw_Dashed_Line pkg. def. 58
 Draw_Line pkg. def. 58
 Draw_Rectangle pkg. def. 58
 Draw_Rectangle_Builtin pkg. def. 58
 Draw_Text pkg. def. 58
 dynamic 2
 Edit pkg. def. 44
 Edit_And_Interpret pkg. def. 45
 Edit_File pkg. def. 44
 End_Of_File pkg. def. 32
 End_Of_Line pkg. def. 32
 End_Of_Page pkg. def. 32
 environment customize and tailor 12
 environment initialization 4
 environment 2
 error messages 123
 evolution 16
 Example of Interactive Ada 130
 Example of Interfacing with Host OS 134
 Example of Manipulating Ada Components 131
 Example of Xt 117
 Example of multiple line input 4

- example of nesting commands 15
- example of *Form* parameters and overload-
ing 9
- example startup.ace file 10
- Execute pkg. def. 47
- execute ACE 5
- execution of ACE 4
- existing environments 8
- Exists pkg. def. 41
- exit Ace 3, 111
- exit Ace_Level 3
- exit Ace_Main 3
- exit 3
- File Management 32
- file object 14
- file system 7, 16, 19, 41
- Form parameter 8, 14
- Form pkg. def. 32, 43, 48
- general file 16
- Get pkg. def. 33, 35, 72-73, 112
- Get_File pkg. def. 44
- Get_Line pkg. def. 33
- Help pkg. def. 50
- help 7, 19, 50
- hierarchical command environment 16
- home directory 9
- Home_Directory pkg. def. 43
- Host Os 19, 38
- Host pkg. def. 38
- host environment support tools 17
- host operating system 7, 17, 19, 38
- Hp Widgets 20, 84, 109
- incomplete command 4
- information hiding 12-13
- inheritance 16
- input and output 3, 14
- interactive 1-2
- interfaces 16
- Interlisp 1
- Interpret pkg. def. 37
- Interpret 9
- interpreted 7, 9, 16-17
- Interpret_File pkg. def. 37
- Interpret_File 12
- Intrinsics 20, 66
- Invoke_Process pkg. def. 59
- Io Exceptions 18, 29
- Is_Open pkg. def. 32, 43, 48
- Key Bindings 19, 23, 51, 119
- Length pkg. def. 36
- library files 8, 14, 23
- Line Counter 19
- Line control 32
- Line length 32
- Line pkg. def. 33
- Line_Length pkg. def. 32
- List Mode 35
- List pkg. def. 39, 43-44, 49, 51
- List 9
- List_Breakpoints pkg. def. 39
- List_File pkg. def. 44
- List_Symbol_Table pkg. def. 39
- locality 12-13
- logical grouping 14
- Low Level Io 18, 30
- Main_Prompt 4, 7
- Make Bindings 119
- Make_Binding pkg. def. 52
- Make_Xt_String pkg. def. 67
- Manipulate Scope 19, 38
- Measure_Mccabe_Complexity pkg. def. 61
- Method of Execution 35
- Mode pkg. def. 32, 48
- modularity 12-13
- Month pkg. def. 30
- Name pkg. def. 32, 43, 48
- nested subprograms and packages 14
- New_Line pkg. def. 32
- New_Page pkg. def. 32
- Null_Caddr_T pkg. def. 66
- Null_Widget pkg. def. 66
- Null_Widget_Class pkg. def. 66
- Null_Xrm_Option_List pkg. def. 67
- Null_Xt_Arg_List pkg. def. 67
- Object Lister 19, 50
- object-oriented design 13
- Objects 2-3, 5, 18, 24
- observe_window.icn 5
- Off 5-7
- On 5-7

Open pkg. def. 32, 42, 47
 Open 8. 14
 operations 2-3. 13
 overloaded suprograms uniquely identified
 8
 overloading 15
 package ADT 8
 packages 13
 Page control 32
 Page length 32
 Page pkg. def. 33
 Page_Length pkg. def. 32
 pragma Continue_Prompt 7
 pragma Debug 6
 pragma Dump 6
 pragma Echo 6
 pragma List_Statement_Numbers 6
 pragma Main_Prompt 7
 pragma Observe 5
 pragma Trace 6
 pragma 3
 pramga Builtin 7
 pre-compiled 7
 Prefix_To_Cais_Arg_List pkg. def. 60
 Print pkg. def. 44
 Program Objects 19, 45
 Program Text Objects 19, 46
 Program Units 19, 47
 Put pkg. def. 33, 35, 49, 72, 112
 Put_Cais_File_Node_Host_Name pkg. def.
 60
 Put_Current_Cais_Node pkg. def. 59
 Put_Current_Directory pkg. def. 43
 Put_File pkg. def. 44
 Put_Line pkg. def. 33
 Put_Time pkg. def. 63
 rapid prototyping 8
 Rename pkg. def. 41
 Rename 16
 Renamed Xlib Types 20, 65
 Reset pkg. def. 32, 42, 57
 runtime efficiency 17
 Seconds pkg. def. 30
 session 3
 Set_Ace_Input pkg. def. 35

Set_Ace_Input 3
 Set_Ace_Output pkg. def. 35
 Set_Break pkg. def. 39
 Set_Col pkg. def. 32
 Set_Current_Cais_Node pkg. def. 59
 Set_Directory pkg. def. 43
 Set_Input pkg. def. 32
 Set_Line pkg. def. 33
 Set_Line_Length pkg. def. 32
 Set_Output pkg. def. 32
 Set_Page_Length pkg. def. 32
 Set_Up pkg. def. 62
 Skip_Line pkg. def. 32
 Skip_Page pkg. def. 32
 Slice pkg. def. 36
 Smalltalk 1
 software engineering 12
 Spawn_Process pkg. def. 59
 Split pkg. def. 30
 standard packages 23
 Standard 18, 23-24
 standard's paradigm 14
 standard.ace 23
 Standard_Ace_Input pkg. def. 35
 Standard_Ace_Input 3
 Standard_Ace_Output pkg. def. 35
 Standard_Ace_Output 3
 Standard_Input pkg. def. 32
 Standard_Output pkg. def. 32
 Stars Tools 19. 61
 startup.ace 4, 9, 18
 Start_Xt pkg. def. 111
 statement database 37
 statement number 8
 Statements 2, 5
 Step pkg. def. 39
 String Io 33
 Strings 18, 28, 36
 strong typing 15
 subprogram ADT 8
 Subprograms 5, 13
 Sun workstation 5
 system decomposition 13
 System 18
 system-dependent 8

technology 16
 Temporary_Name pkg. def. 41
 Test_Case_Generator pkg. def. 62
 Test_Comparator pkg. def. 62
 Test_Procedures_Generator pkg. def. 62
 Test_Results_Analyzer pkg. def. 62
 Test_Updater pkg. def. 62
 Text Files 16
 Text Objects 19, 44
 Text_Io 3, 8, 31
 Text_Width pkg. def. 64
 Time_Of pkg. def. 30
 traditional view 7
 Undelete pkg. def. 38
 uniformity 14
 visibility 2
 Widget Package 20, 67
 widget 109
 Window Draw Routines 19, 58
 Window Objects 14, 19, 56, 109
 window environment 5
 windowing 8, 19, 23, 56
 windows.ace 23
 X Window System 5
 X Windows 20, 23, 64
 Xt Stringdefs 20
 Xt toolkit 63, 109
 Xt_Add_Actions pkg. def. 75
 Xt_Add_Actions 114
 Xt_Add_Callback pkg. def. 73-74, 113
 Xt_App_Next_Event pkg. def. 75
 Xt_Arg_List 114
 Xt_Augment_Translations pkg. def. 75
 Xt_Create_Managed_Widget pkg. def. 74
 Xt_Create_Widget pkg. def. 74
 Xt_Default_App_Context pkg. def. 75
 Xt_Destroy_Widget pkg. def. 74
 Xt_Dispatch_Event pkg. def. 67, 75
 Xt_Get_Value pkg. def. 73
 Xt_Initialize pkg. def. 67, 110
 Xt_Main_Loop pkg. def. 74
 Xt_Override_Translations pkg. def. 75
 Xt_Parse_Translation_Table pkg. def. 75
 Xt_Realize_Widget pkg. def. 74
 Xt_Set_Arg pkg. def. 68-72, 77-78, 96-107, 109
 Xt_Set_Arg 112
 Xt_Set_Values pkg. def. 73
 Xw_Arrow_Widget_Class pkg. def. 92
 Xw_Ascii_Sink_Create pkg. def. 95
 Xw_Bulletin_Board_Widget_Class pkg. def. 92
 Xw_Bulletin_Widget_Class pkg. def. 92
 Xw_Button_Widget_Class pkg. def. 92
 Xw_Cascade_Widget_Class pkg. def. 92
 Xw_Disk_Source_Create pkg. def. 95
 Xw_Disk_Source_Destroy pkg. def. 95
 Xw_Form_Widget_Class pkg. def. 92
 Xw_Image_Edit_Widget_Class pkg. def. 92
 Xw_Listrow_Col_Widget_Class pkg. def. 92
 Xw_List_Widget_Class pkg. def. 92
 Xw_Manager_Widget_Class pkg. def. 92
 Xw_Menubutton_Widget_Class pkg. def. 92
 Xw_Menumgr_Widget_Class pkg. def. 92
 Xw_Menupane_Widget_Class pkg. def. 92
 Xw_Menu_Button_Widget_Class pkg. def. 92
 Xw_Menu_Sep_Widget_Class pkg. def. 92
 Xw_Move_Focus pkg. def. 95
 Xw_Panel_Widget_Class pkg. def. 92
 Xw_Popupmgr_Widget_Class pkg. def. 92
 Xw_Popup_Mgr_Widget_Class pkg. def. 92
 Xw_Primitive_Widget_Class pkg. def. 92
 Xw_Push_Button_Widget_Class pkg. def. 92
 Xw_Row_Col_Widget_Class pkg. def. 92
 Xw_Sash_Widget_Class pkg. def. 92
 Xw_Scrollbar_Widget_Class pkg. def. 92
 Xw_Scrolled_Window_Widget_Class pkg. def. 92
 Xw_Scroll_Bar_Widget_Class pkg. def. 92
 Xw_Sraster_Widget_Class pkg. def. 93
 Xw_Statictext_Widget_Class pkg. def. 93
 Xw_Static_Raster_Widget_Class pkg. def. 93
 Xw_Static_Text_Widget_Class pkg. def. 93
 Xw_String_Source_Create pkg. def. 95
 Xw_String_Source_Destroy pkg. def. 95
 Xw_Swindow_Widget_Class pkg. def. 93

Xw_Textedit_Widget_Class pkg. def. 93
Xw_Text_Clear_Buffer pkg. def. 93
Xw_Text_Copy_Buffer pkg. def. 93
Xw_Text_Copy_Selection pkg. def. 93
Xw_Text_Edit_Widget_Class pkg. def. 93
Xw_Text_Get_Insert_Pos pkg. def. 95
Xw_Text_Get_Last_Pos pkg. def. 94
Xw_Text_Get_Selection_Pos pkg. def. 94
Xw_Text_Insert pkg. def. 94
Xw_Text_Read_Sub_String pkg. def. 94
Xw_Text_Redraw pkg. def. 94
Xw_Text_Replace pkg. def. 94
Xw_Text_Set_Insert_Pos pkg. def. 95
Xw_Text_Set_Selection pkg. def. 94
Xw_Text_Set_Source pkg. def. 95
Xw_Text_Unset_Selection pkg. def. 94
Xw_Text_Update pkg. def. 94
Xw_Titlebar_Widget_Class pkg. def. 93
Xw_Title_Bar_Widget_Class pkg. def. 93
Xw_Toggle_Widget_Class pkg. def. 93
Xw_Valuator_Widget_Class pkg. def. 93
Xw_Work_Space_Widget_Class pkg. def. 93
X_Text_Width pkg. def. 75
Year pkg. def. 30